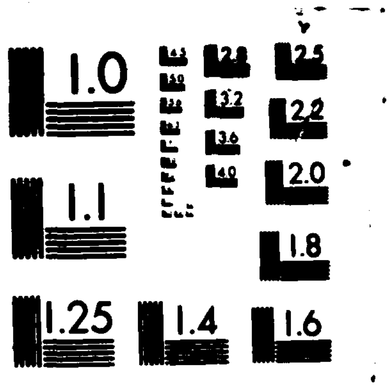
AD-A183 935 PRODUCTION SCHEDULING OF SEQUENCED TAPES FOR PRINTED CIRCUIT PACK ASSEMBLY(U) ARMY MILITARY PERSONNEL CENTER ALEXANDRIA VA PROGRAM ANALYSIS OFFICE S J RAWLICK 1/2

UNCLASSIFIED 09 JUL 87 F/G 9/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Production Scheduling of Sequenced Tapes for Printed Circuit Pack Assembly

Steven J. Rawlick, CPT
HQDA, MILPERCEN (DAPC-OPA-E)
200 Stovall Street
Alexandria, VA 22332

Final report 9 July, 1987

A thesis submitted to North Carolina State University in Raleigh, North Carolina in partial fulfillment of the requirements for the degree of Master of Science.

```
IF(UNION .EQ. O .AND. COMP .EQ. 1) THEN
   DIFFER = DIFFER + 1
ENDIF
C
RETURN
END
C
C *************************************************************************
```

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE	*Form Approved OMB No 0704-0188 Exp Date Jun 30, 1986*
1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Production Scheduling of Sequenced Tapes for Printed Circuit Pack Assembly	5. MONITORING ORGANIZATION REPORT NUMBER(S) Final Report 9 July 87
6a. NAME OF PERFORMING ORGANIZATION / 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION CPT Steven J. Rawlick
6c. ADDRESS *(City, State, and ZIP Code)*	7b. ADDRESS *(City, State, and ZIP Code)* 3433 Mt. Burnside Way Woodbridge, VA 22192
8a. NAME OF FUNDING/SPONSORING ORGANIZATION / 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Student, HQDA, MILPERCEN (DAPC-OPA-E), 200 Stovall St. Alexandria VA 22332
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE *(Include Security Classification)* HQDA, MILPERCEN, ATTN: DAPC-OPA-E, 200 Stovall St, Alexandria, VA 22332	
12. PERSONAL AUTHOR(S) 9 July 87	
13a. TYPE OF REPORT Final / 13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT *(Year, Month, Day)* / 15. PAGE COUNT 137
16. SUPPLEMENTARY NOTATION Approved for public release; distribution unlimited.	
17. COSATI CODES: FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)* University thesis
19. ABSTRACT *(Continue on reverse if necessary and identify by block number)* see enclosed abstract	
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE *(Include Area Code)* / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

INSTRUCTIONS FOR PREPARATION OF REPORT DOCUMENTATION PAGE

GENERAL INFORMATION

The accuracy and completeness of all information provided in the DD Form 1473, especially classification and distribution limitation markings, are the responsibility of the authoring or monitoring DoD activity.

Because the data input on this form will be what others will retrieve from DTIC's bibliographic data base or may determine how the document can be accessed by future users, care should be taken to have the form completed by knowledgeable personnel. For better communication and to facilitate more complete and accurate input from the originators of the form to those processing the data, space has been provided in Block 22 for the name, telephone number, and office symbol of the DoD person responsible for the input cited on the form.

All information on the DD Form 1473 should be typed.

Only information appearing on or in the report, or applying specifically to the report in hand, should be reported. If there is any doubt, the block should be left blank.

Some of the information on the forms (e.g., title, abstract) will be machine indexed. The terminology used should describe the content of the report or identify it as precisely as possible for future identification and retrieval.

NOTE: **Unclassified abstracts and titles describing classified documents may appear separately from the documents in an unclassified context, e.g., in DTIC announcement bulletins and bibliographies. This must be considered in the preparation and marking of unclassified abstracts and titles.**

The Defense Technical Information Center (DTIC) is ready to offer assistance to anyone who needs and requests it. Call Data Base Input Division, Autovon 284-7044 or Commercial (202) 274-7044.

SECURITY CLASSIFICATION OF THE FORM

In accordance with DoD 5200.1-R, Information Security Program Regulation, Chapter IV Section 2, paragraph 4-200, classification markings are to be stamped, printed, or written at the top and bottom of the form in capital letters that are larger than those used in the text of the document. See also DoD 5220.22-M, Industrial Security Manual for Safeguarding Classified Information, Section II, paragraph 11a(2). This form should be unclassified, if possible.

SPECIFIC BLOCKS

Block 1a. Report Security Classification: Designate the highest security classification of the report. (See DoD 5220.1-R, Chapters I, IV, VII, XI, Appendix A.)

Block 1b. Restricted Marking: Enter the restricted marking or warning notice of the report (e.g., CNWDI, RD, NATO).

Block 2a Security Classification Authority: Enter the commonly used markings in accordance with DoD 5200.1-R, Chapter IV, Section 4, paragraph 4-400 and 4-402 Indicate classification authority.

Block 2b. Declassification / Downgrading Schedule: Indicate specific date or event for declassification or the notation, "Originating Agency Determination Required" or "OADR." Also insert (when applicable) downgrade to ______________ on ______________ (e.g., Downgrade to Confidential on 6 July 1983). (See also DoD 5220.22-M, Industrial Security Manual for Safeguarding Classified Information, Appendix II.)

NOTE: **Entry must be made in Blocks 2a and 2b except when the original report is unclassified and has never been upgraded.**

Block 3. Distribution/Availability Statement of Report: Insert the statement as it appears on the report. If a limited distribution statement is used, the reason must be one of those given by DoD Directive 5200 20, Distribution Statements on Technical Documents, as supplemented by the 18 OCT 1983 SECDEF Memo, "Control of Unclassified Technology with Military Application." The Distribution Statement should provide for the broadest distribution possible within limits of security and controlling office limitations

Block 4. Performing Organization Report Number(s): Enter the unique alphanumeric report number(s) assigned by the organization originating or generating the report from its research and whose name appears in Block 6 These numbers should be in accordance with ANSI STD 239 23-74, "American National Standard Technical Report Number." If the Performing Organization is also the Monitoring Agency, enter the report number in Block 4

Block 5 Monitoring Organization Report Number(s): Enter the unique alphanumeric report number(s) assigned by the Monitoring Agency This should be a number assigned by a DoD or other government agency and should be in accordance with ANSI STD 239 23-74 If the Monitoring Agency is the same as the Performing Organization, enter the report number in Block 4 and leave Block 5 blank

Block 6a Name of Performing Organization: For in-house reports, enter the name of the performing activity For reports prepared under contract or grant, enter the contractor or the grantee who generated the report and identify the appropriate corporate division, school, laboratory, etc., of the author

Block 6b Office Symbol: Enter the office symbol of the Performing Organization

Block 6c Address Enter the address of the Performing Organization List city, state, and ZIP code

Block 7a Name of Monitoring Organization: This is the agency responsible for administering or monitoring a project, contract, or grant If the monitor is also the Performing Organization, leave Block 7a blank In the case of joint sponsorship, the Monitoring Organization is determined by advance agreement It can be either an office, a group, or a committee representing more than one activity, service, or agency

Block 7b Address: Enter the address of the Monitoring Organization Include city, state, and ZIP code

Block 8a Name of Funding/Sponsoring Organization Enter the full official name of the organization under whose immediate funding the document was generated, whether the work was done in-house or by contract If the Monitoring Organization is the same as the Funding Organization, leave 8a blank

Block 8b Office Symbol Enter the office symbol of the Funding/Sponsoring Organization

Block 8c Address Enter the address of the Funding/Sponsoring Organization Include city state and ZIP code

Block 9. Procurement Instrument Identification Number: For a contractor grantee report, enter the complete contract or grant number(s) under which the work was accomplished. Leave this block blank for in-house reports.

Block 10. Source of Funding (Program Element, Project, Task Area, and Work Unit Number(s): These four data elements relate to the DoD budget structure and provide program and/or administrative identification of the source of support for the work being carried on. Enter the program element, project, task area, work unit accession number, or their equivalents which identify the principal source of funding for the work required. These codes may be obtained from the applicable DoD forms such as the DD Form 1498 (Research and Technology Work Unit Summary) or from the fund citation of the funding instrument. If this information is not available to the authoring activity, these blocks should be filled in by the responsible DoD Official designated in Block 22. If the report is funded from multiple sources, identify only the Program Element and the Project, Task Area, and Work Unit Numbers of the principal contributor.

Block 11. Title: Enter the title in Block 11 in initial capital letters exactly as it appears on the report. Titles on all classified reports, whether classified or unclassified, must be immediately followed by the security classification of the title enclosed in parentheses. A report with a classified title should be provided with an unclassified version if it is possible to do so without changing the meaning or obscuring the contents of the report. Use specific, meaningful words that describe the content of the report so that when the title is machine-indexed, the words will contribute useful retrieval terms.

If the report is in a foreign language and the title is given in both English and a foreign language, list the foreign language title first, followed by the English title enclosed in parentheses. If part of the text is in English, list the English title first followed by the foreign language title enclosed in parentheses. If the title is given in more than one foreign language, use a title that reflects the language of the text. If both the text and titles are in a foreign language, the title should be translated, if possible, unless the title is also the name of a foreign periodical. Transliterations of often used foreign alphabets (see Appendix A of MIL-STD-847B) are available from DTIC in document AD-A080 800.

Block 12. Personal Author(s): Give the complete name(s) of the author(s) in this order: last name, first name, and middle name. In addition, list the affiliation of the authors if it differs from that of the performing organization.

List all authors. If the document is a compilation of papers, it may be more useful to list the authors with the titles of their papers as a contents note in the abstract in Block 19. If appropriate, the names of editors and compilers may be entered in this block.

Block 13a. Type of Report: Indicate whether the report is summary, final, annual, progress, interim, etc.

Block 13b. Time Covered: Enter the inclusive dates (*year, month, day*) of the period covered, such as the life of a contract in a final contractor report.

Block 14. Date of Report: Enter the year, month, and day, or the year and the month the report was issued as shown on the cover.

Block 15. Page Count: Enter the total number of pages in the report that contain information, including cover, preface, table of contents, distribution lists, partial pages, etc. A chart in the body of the report is counted even if it is unnumbered.

Block 16. Supplementary Notation: Enter useful information about the report in hand, such as: "Prepared in cooperation with ...," "Translation at (or by) ...," "Symposium ...." If there are report numbers for the report which are not noted elsewhere on the form (such as internal series numbers or participating organization report numbers) enter in this block.

Block 17. COSATI Codes. This block provides the subject coverage of the report for announcement and distribution purposes. The categories are to be taken from the "COSATI Subject Category List" (DoD Modified), Oct 65, AD-624 000. A copy is available on request to any organization generating reports for DoD. At least one entry is required as follows.

Field - to indicate subject coverage of report

Group - to indicate greater subject specificity of information in the report

Sub-Group - if specificity greater than that shown by Group is required, use further designation as the numbers after the period (.) in the Group breakdown. Use only the designation provided by AD-624 000.

Example: The subject "Solid Rocket Motors" is Field 21, Group 08, Subgroup 2 (page 32, AD-624 000).

Block 18. Subject Terms: These may be descriptors, keywords, posting terms, identifiers, open-ended terms, subject headings, acronyms, code words, or any words or phrases that identify the principal subjects covered in the report, and that conform to standard terminology and are exact enough to be used as subject index entries. Certain acronyms or "buzz words" may be used if they are recognized by specialists in the field and have a potential for becoming accepted terms. "Laser" and "Reverse Osmosis" were once such terms.

If possible, this set of terms should be selected so that the terms individually and as a group will remain UNCLASSIFIED without losing meaning. However, priority must be given to specifying proper subject terms rather than making the set of terms appear "UNCLASSIFIED." Each term on classified reports must be immediately followed by its security classification, enclosed in parentheses.

For reference on standard terminology the "DTIC Retrieval and Indexing Terminology" DRIT-1979, AD-A068 500, and the DoD "Thesaurus of Engineering and Scientific Terms (TEST) 1968, AD-672 000, may be useful.

Block 19. Abstract: The abstract should be a pithy, brief (preferably not to exceed 300 words), factual summary of the most significant information contained in the report. However, since the abstract may be machine-searched, all specific and meaningful words and phrases which express the subject content of the report should be included, even if the word limit is exceeded.

If possible, the abstract of a classified report should be unclassified and consist of publicly releasable information (Unlimited), but in no instance should the report content description be sacrificed for the security classification.

NOTE: **An unclassified abstract describing a classified document may appear separately from the document in an unclassified context e.g., in DTIC announcement or bibliographic products. This must be considered in the preparation and marking of unclassified abstracts.**

For further information on preparing abstracts, employing scientific symbols, verbalizing, etc., see paragraphs 2.1(n) and 2.3(b) in MIL-STD-847B.

Block 20. Distribution / Availability of Abstract. This block must be completed for all reports. Check the applicable statement: "unclassified / unlimited," "same as report," or, if the report is available to DTIC registered users "DTIC users."

Block 21. Abstract Security Classification. To ensure proper safeguarding of information, this block must be completed for all reports to designate the classification level of the entire abstract. For CLASSIFIED abstracts, each paragraph must be preceded by its security classification code in parentheses.

Block 22a,b,c. Name, Telephone and Office Symbol of Responsible Individual. Give name, telephone number, and office symbol of DoD person responsible for the accuracy of the completion of this form.

determined. The derived computational results indicate that a heuristic approach is computationally efficient, although only a suboptimal solution is achieved.

ABSTRACT

RAWLICK, STEVEN JOHN. Production Scheduling of Sequenced Tapes for Printed Circuit Pack Assembly. (Under the direction of Yahya Fathi.)

A sequencing machine, or simply, a sequencer, is a piece of equipment used in the electronic assembly industry to produce sequenced reel-packaged tapes of axial leaded components for different types of Printed Circuit Packs (PCPs). Due to the limited number of dispensing heads available on sequencers, the relatively large number of component types competing for these heads, and the diversity of the component type requirements of different types of PCPs, efficient scheduling of these machines is usually not a simple task.

Fathi and Taheri [1986] developed a mathematical model pertaining to one variation of the sequenced tape production scheduling problem. They employ a strategy aimed at providing an optimal solution by totally eliminating all change-over time between consecutive runs on the available sequencers. Their integer programming model is discussed, and the performance of their model is examined. Test results provide evidence that the particular variation of the sequenced tape production scheduling problem which they confront is intractable through branch-and-bound techniques due to lengthy computation times.

A heuristic approach to solve another variation of the sequenced tape production scheduling problem is presented. Three different heuristic procedures employing a specific solution strategy are developed. The algorithm followed by the three heuristic procedures is described, and the relative merits of three procedures are empirically

PRODUCTION SCHEDULING OF SEQUENCED TAPES FOR PRINTED CIRCUIT PACK ASSEMBLY

by

STEVEN JOHN RAWLICK

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

DEPARTMENT OF OPERATIONS RESEARCH

Raleigh

1987

APPROVED BY:

[illegible] [illegible]

[illegible] [illegible]

Chairman of Advisory Committee

ACKNOWLEDGEMENTS

I am most grateful to Dr. Yahya Fathi, the Chairman of my Advisory Committee. His knowledge and patient guidance have had a large impact on my capacity to undertake this project and derive meaningful results.

I owe a special thanks to William G. Ferrell and Dr. Salah E. Elmaghraby. Bill Ferrell provided invaluable assistance, and spotted possible obstacles which permitted me to focus my research in a much more direct manner. Dr. Elmaghraby gave me the opportunity to continually improve in the area of operations research.

Most importantly, I am deeply indebted to my wife, Ginny, whose tireless support and understanding were instrumental in allowing me to dedicate the time and effort required for this thesis. I also want to acknowledge my parents and my daughter, who help to make everything worthwhile.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER 1

INTRODUCTION

The electronic assembly industry, as with most technical industries, is unquestionably devoted to computerized, vastly efficient production operations. A particular production item in huge demand is a Printed Circuit Pack (PCP). Broadly speaking, a PCP is assembled via an automated system that inserts components into a Printed Circuit Board (PCB). The components themselves are pre-packaged in a specific sequence on a large reel of tape which successively feeds the components to the PCB (which, in reality, is a blank PCP). The resulting product of this computer-controlled insertion process is the completed Printed Circuit Pack.

Printed Circuit Packs are distinguishable from each other by their component composition. PCPs differ in the types, and the number of each type of component required for their production. Consequently, in order to assemble a particular PCP, a specifically sequenced reel-packaged tape containing all of the components required to produce that PCP must be prepared in advance. However, components initially are reel-packaged by type only. In other words, each reel contains a large number of identical components. Given the task of producing a sequenced reel-packaged tape, then, the requisite components must be detached from the various component tapes on hand and spliced in a specific sequence onto one blank receiving tape.

This detachment/splicing operation is commonly referred to in the electronic assembly industry as sequencing and packaging. A condensed description of a sequencing/packaging operation and PCP production process is extracted from a technical report by Fathi and Taheri [1986, p. 3]:

> A Variable Center Distance automatic insertion machine (VCD) is a piece of equipment used in the electronic assembly industry to insert axial leaded components into Printed Circuit Boards (PCBs) to produce Printed Circuit Packs (PCPs). Sequenced and reel-packaged components are loaded onto a VCD, as input material, while a numerically controlled pattern program directs the operation of insertion of these components into the PCB. Therefore, for each type of PCP, the proper set of components must be sequenced according to the requirements of the pattern program and packaged on tapes (reel-packaged) prior to their use on a VCD. Typically, this operation (sequencing and packaging) is performed on a machine known as a Sequencing Machine, or simply, a Sequencer.
>
> A PCP assembly plant typically has several such sequencers, each of which represents a substantial investment. Efficient scheduling of these sequencers is a common concern of the management from both the viewpoint of equipment utilization and that of the throughput of the insertion process. Because of the combinatorial nature of the problems, efficient scheduling of the sequencers is not a simple task.

The problem alluded to above is known as a sequenced tape production scheduling problem. This problem has several variations directly attributable to factors such as the number and type of PCPs to be produced, and the resources available to produce them. The thrust of this paper encompasses two distinct, yet related variations. The first variation discussed is termed a sequencer assignment problem, which is addressed by Fathi and Taheri in their report. Basically, a sequencer assignment problem represents a situation in which sequencing and packaging resources are, in a sense, unconstrained. The issue is solving the problem so as to optimally utilize as few of the available resources as possible. Fathi and Taheri present a mathematical programming model that can be used to resolve some of the difficulties involved when dealing with a sequencer assignment problem.

In this thesis, an effort is made to examine that mathematical model and its performance, but the bulk of the research was influenced by their closing remarks concerning a related problem. This variation is termed a sequencer scheduling/assignment problem, which represents a situation where sequencing and packaging resources are, in fact, constrained. The reader will note that the nomenclature of the two variations is similar in that they both contain the words 'sequencer assignment problem', and they differ only by the exclusion/inclusion of the word 'scheduling'. By extension, it is fair to assume that these two problems are different because the 'scheduling' of something (yet to be explained) is not important in the first, but is critical in the second. Permit us to digress at this point to generally characterize the 'sequencer assignment problem' and illustrate why 'scheduling' is disregarded in one case, yet essential in the other.

Consider an environment where one sequencer is available. This sequencer is equipped with a fixed number of dispensing heads. Reels of components are mounted on these dispensing heads so that individual components may be detached onto a blank tape to produce a sequenced reel-packaged tape, or sequenced tape. The different component types required to produce a single PCP, or pack type, are known in advance. Therefore, the requisite component tapes are mounted onto various sequencer dispensing heads so that the sequenced tape corresponding to the particular pack type may be produced.

This sequencing and packaging operation seems straightforward. However, the environment becomes congested when different pack types need to be produced. The possibility exists that the total number of component types needed to produce all of the different pack types exceeds the fixed number of sequencer dispensing heads currently available. If this situation occurs, it is inevitable that at some point during the sequenced tape production process, the sequencing machine will have to halt so that the necessary, unmounted component tapes may be loaded, causing unneeded component tapes to be removed. Once this change-over operation is completed, the sequencing and packaging operation resumes. The only alternative to this situation is to purchase extra dispensing heads to affix to the sequencer in order to accommodate all of the different component tapes. Given this alternative, which requires a favorable decision at management level, the sequencer again would be capable of producing all of its assigned sequenced tapes without halting.

Halting the sequencer during the production process of the sequenced tapes is the key issue. If the sequencer can produce all of its assigned sequenced tapes without change-overs, then the only set-up task is to load the appropriate component tapes onto the dispensing heads prior to production start-up. This situation, accurately portrayed as the sequencer assignment problem, does not require a sequenced tape production schedule. In other words, the order of production of the different types of sequenced tapes, corresponding to the different pack types, does not affect their total production time. This is true, because under the assumption that no change-overs will occur, there will be no set-up time between production of consecutive types of sequenced tapes. It follows that all schedules (orders of production) result in the same total production time, hence are equally good.

This, however, is certainly not the case when the sequencer cannot produce all of its assigned sequenced tapes without halting. It may be cost effective for management to tolerate this situation depending on the duration of the sequencer halts, and the costs incurred to purchase extra dispensing heads. Naturally, the sequenced tapes must be assigned to the sequencer for production, but the order of production of the sequenced tapes is now vitally important. The logic behind this stems from the fact that some pack types may have more in common with others in terms of component composition. A judicious production schedule might group common pack types together. By establishing this sort of relationship, ideally then, the sequencer would produce as many sequenced tapes as possible before halting to take on other needed component tapes. The production schedule also dictates which specific component tapes must be loaded onto the dispensing heads, and when they are to be loaded. This situation is fittingly portrayed as the sequencer scheduling/assignment problem. It is reasonable to assume that shrewd scheduling might result in shorter, and possibly fewer sequencer halts. This thought process is very much in line with industry's desire to maintain vastly efficient production operations.

The sequencer scheduling/assignment problem referred to in the closing remarks by Fathi and Taheri is a much more complex problem than the sequencer assignment problem, which they treat in great detail.

The subject matter of this thesis is inspired directly from their technical report. The goals of this research are twofold:

1) Test their mathematical model of the sequencer assignment problem to develop an insight into its performance in a typical problem environment. A test phase would hopefully enable conclusions to be drawn concerning the model's efficiency and possibly detect particular aspects inherent in the model that might lead to other areas of research.

2) Study the more realistic sequencer scheduling/assignment problem, which would be created by management's decision not to purchase additional dispensing heads, but instead, to tolerate a certain amount of change-over time between consecutive runs on the sequencer. This problem, spurred by a managerial attitude of, in effect, 'Work with the resources currently available and prepare the corresponding sequenced tapes in accordance with the specified pack type requirements', necessitates a mathematical model altogether different from the Fathi/Taheri model.

Due to the nature of the results derived during the test phase of the Fathi/Taheri model, it became readily apparent that the two aforementioned goals actually lead to one and the same objective, which constitutes the basis for this research. This objective is the determination of a production schedule of sequenced tapes that minimizes the total change-over time between consecutive runs on the sequencers for a given set of pack types and their associated volume requirements. An implied task in this study is to ensure that the distribution of total workload among the available sequencers is relatively even throughout the sequenced tape production process.

As alluded to earlier, the technical report presented by Fathi and Taheri is the cornerstone of this study. In their report, they offer a very condensed description of the operations performed by Variable Center Distance automatic insertion machines (VCDs), and the operations performed by sequencing machines. The thrust of their report centers on the sequencer assignment problem, and the mathematical programming model developed to provide a solution for that problem.

VCD operations are very complex in nature, and are the subject of intense scrutiny in the electronic assembly industry. In a working paper by Saboo et al. [1986], electronic assembly operations with respect to Printed Circuit Boards are discussed, and a detailed explanation of VCD operations is provided. The operation of sequencers is an integral portion of the overall PCP assembly process, and as such, requires intense scrutiny as well. For a description of the sequencer operations, see the Sequencer User's Manual, by Universal Instruments Corporation [1986].

The mathematical programming model developed by Fathi and Taheri is a type of model frequently used for problem solving and decision making in production systems. One category of mathematical programming model is known as linear programming. For a review of linear programming, see Ozan [1986] or Murty [1983].

An Integer Programming model is a Linear Programming model with an extra requirement that some (or all) of its variables must be integer valued. The model designed by Fathi and Taheri is an integer programming model. Salkin [1975] discusses integer programming in detail, and describes various techniques utilized to formulate and solve such problems.

Chapter 2 defines the sequencer assignment problem and discusses the strategy supporting the mathematical model presented by Fathi and Taheri. The reader is permitted an insight into the model's performance when tested using several sets of typical problem parameters. The necessity of an alternative approach is demonstrated. Chapter 3 discusses the strategy employed to confront the sequencer scheduling/assignment problem. Some measures of goodness by which different heuristic procedures may be compared with each other are described. In Chapter 4, the assumptions used in developing a specific heuristic approach designed to provide a solution to the sequencer scheduling/assignment problem are discussed. This chapter also contains the specific algorithm followed by three different heuristic procedures. Chapter 5 describes the specific aspects of the three heuristic procedures, and provides an example demonstrating a particular procedure performed by the algorithm. Chapter 6 provides computational results, and offers a comparison of the three heuristic procedures and an interpretative discussion of the results. Avenues for further research are presented.

CHAPTER 2

A MATHEMATICAL MODEL FOR LOADING SEQUENCERS

2.1 Problem Background

For any Printed Circuit Pack, or pack type, to be produced, the different component types required for that pack type must be packaged on a reel in the proper sequence. The sequenced reel of component types is placed on a VCD which cuts the components from the reel and inserts them into a Printed Circuit Board. The insertion process is controlled by a pattern program. Therefore, for every pack type that must be produced on a VCD, a sequenced tape of required component types must be previously prepared by a sequencer. The component types that make up the packaged reel are inserted onto the tape in a specific reverse sequenced order, again controlled by a pattern program, so that they will be later inserted into the board correctly by the VCD.

Before delving into the mathematical programming model developed by Fathi and Taheri treating the sequencer assignment problem, it is essential to thoroughly understand the operation of sequencers. Without this understanding, it is utterly impossible to fully grasp the problem at hand, nor is it possible to gain insight into the methods utilized to provide a solution. Consequently, an extracted portion from their report explains the operation of sequencers [1986, p. 4]:

> A Sequencer consists of a set of dispensing heads that cut the components from input reels according to the bill of materials and the insertion pattern. The dispensed components are fed onto a conveyor chain where they are spaced and remounted on an automatically fed tape. Figures 2.1 and 2.2 depict schematic views of a sequencer and the sequencing operation, respectively.

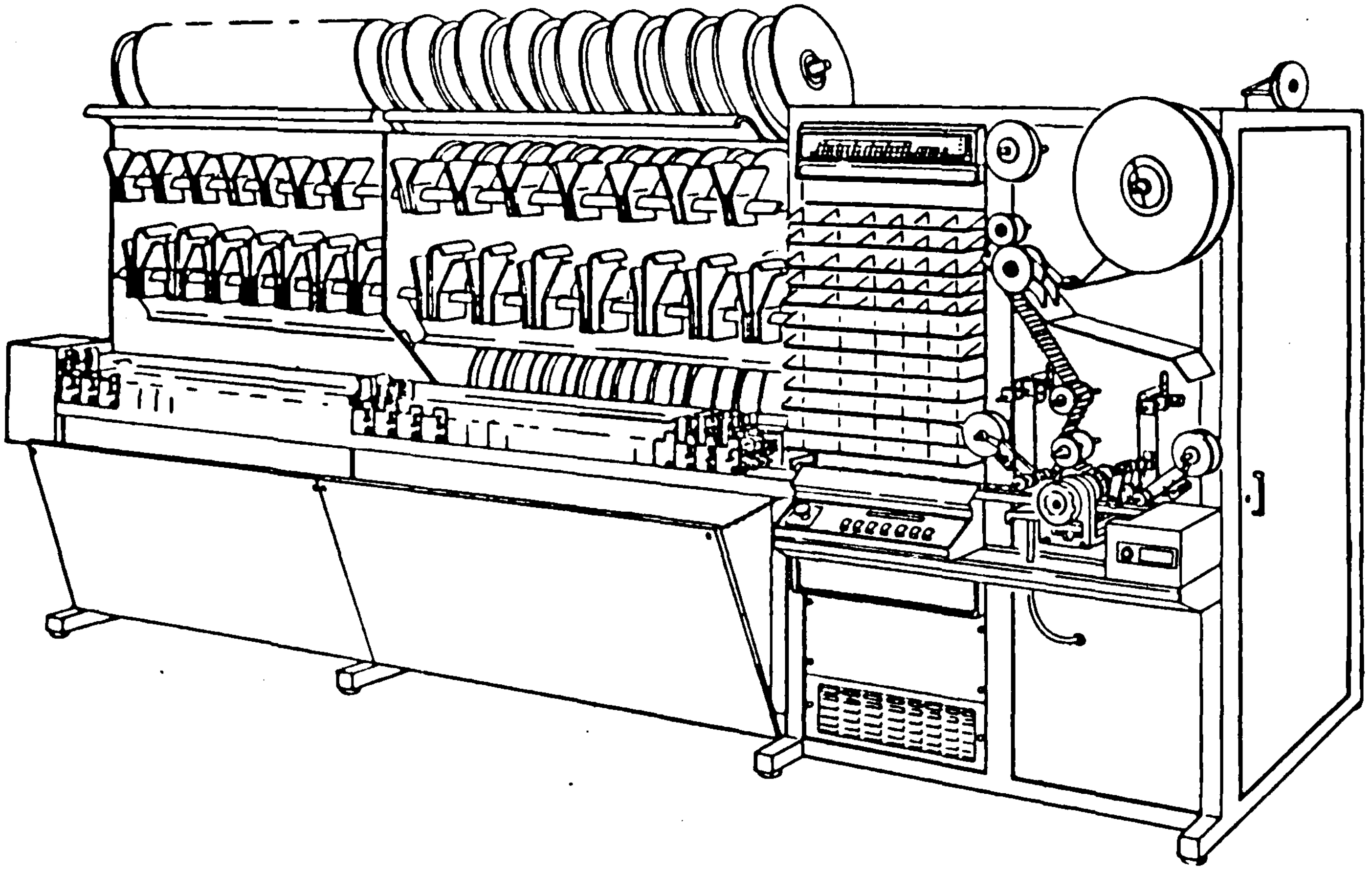


Figure 2.1 The Expandable Component Sequencer
(Reproduced with permission from Universal Instruments Corporation (LTR # EVM - 86-070))

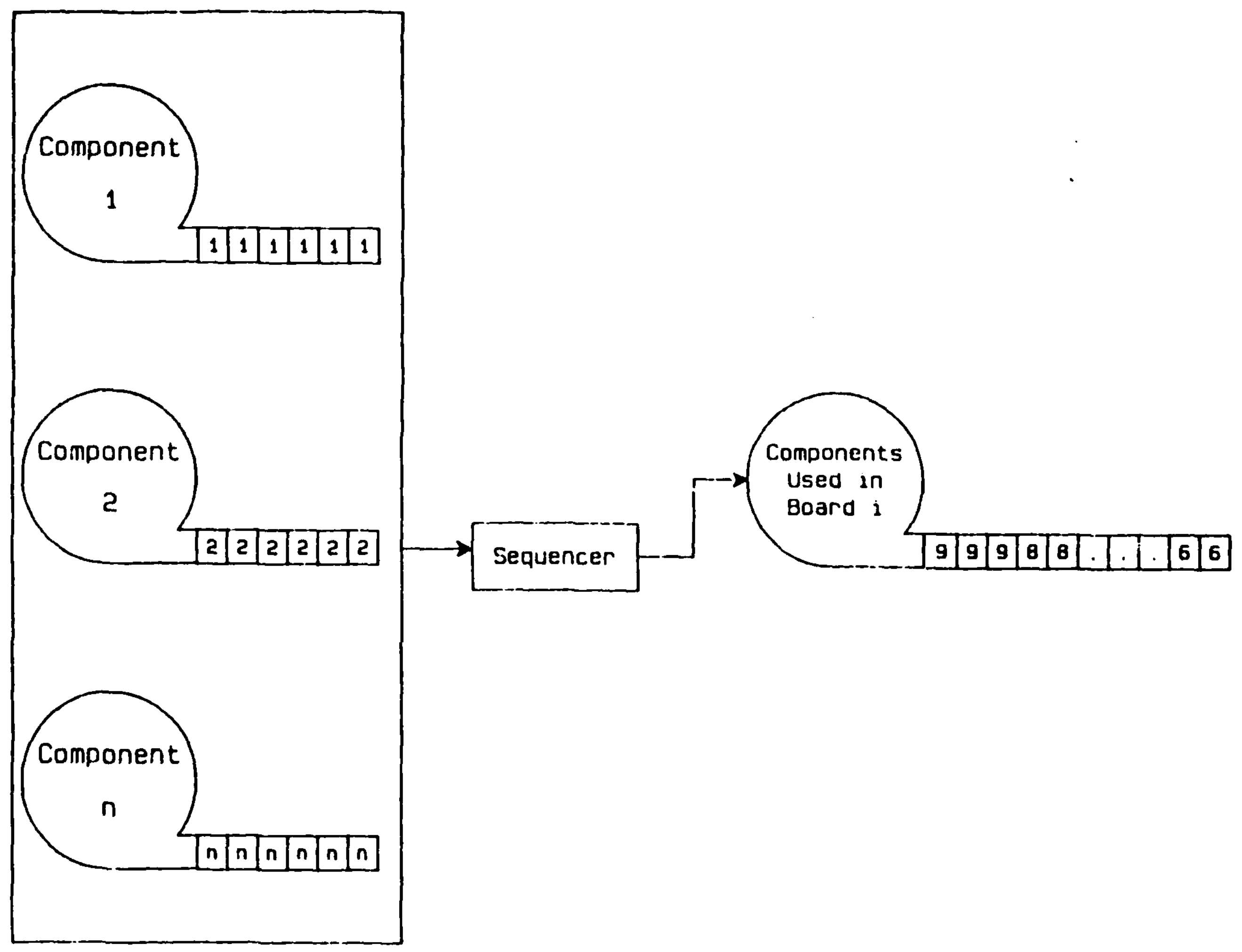


Figure 2.2 A Schematic View of the Sequencing Operation

> Like VCD machines, sequencers are typically run by control programs which are written for specific Printed Circuit Pack (PCP) designs. When developing these control programs, the programmer needs to assume that specific component types are available on specific dispensing heads. Accordingly, prior to operating the sequencer, an operator must verify that these components are actually loaded on the appropriate dispensing heads. This may require unloading some component tapes from the dispensing heads and loading the required set of component tapes in their place. We refer to the time required to perform this unloading/loading operation as the change-over time.
>
> As a general rule, the management usually prefers to streamline the PCP production process by reducing the change-over time between consecutive runs on the sequencers (even if this requires additional capital investment). To achieve this managerial goal, a variety of different strategies could be employed.

They later describe a typical problem environment consisting of 3 to 6 sequencers, 20 to 40 pack types, and anywhere from 200 to 400 different component types. This sequencer assignment problem is combinatorial in nature and can be described thusly.

Throughout the remainder of this thesis, we use the terms <u>pack type</u>, <u>pack</u>, and <u>sequenced tape</u> synonymously, for all intents and purposes. Although sequenced tapes are prepared prior to the production of pack types, this interchangeable relationship is feasible because a pack type and its corresponding sequenced tape require identical component types. Therefore, when a phrase appears such as '. . . a pack type must be produced, and is assigned for production to a sequencer . . .,' it should be clearly understood that the corresponding sequenced tape is actually assigned to the sequencer for production, since sequencers produce sequenced tapes, not pack types.

This conscious abuse of terminology facilitates the comprehension of complex passages and serves to unclutter lengthy explanations.

Each pack type consists of a fixed number of different component types, and each available sequencer has a fixed number of dispensing heads available for mounting component tapes. This is an essential piece of information. It is clearly impossible to commence production of a sequenced tape corresponding to a specific pack type, then halt the sequencing operation in order to load other needed component tapes onto the sequencer. Therefore, we know that if any pack type is assigned to a particular sequencer, then that sequencer is capable of producing the corresponding sequenced tape without halting, since the number of dispensing heads available for mounting component tapes is greater than or at least equal to the number of different component types required to produce that sequenced tape.

Of the 20 to 40 pack types to be produced, some may bear resemblance, while others may be distinctly different. By this, it is inferred that some pack types may require a similar set of different component types, and/or that most of the different component types required for production are common to each. On the contrary, some pack types may have very few component types in common with others, and/or the number of component types required to produce them may differ drastically, possibly up to the maximum number of different component types that any pack type would require.

To capsulize the sequencer loading situation, we assume that within the planning horizon, a PCP assembly plant must produce 20 to 40 pack types, none of which requires more than a fixed number of different component types in order to be completed (for purposes of this paper, the fixed number is set at 100). The plant has 3 to 6 sequencers available for production, and none of the sequencers have more than 100 dispensing heads available (the identical fixed number) to mount component tapes. The total number of different component types needed to produce all of the pack types within the planning horizon will not exceed 400. The ideal solution is a pack type-to-sequencer assignment plan that permits all of the corresponding sequenced tapes to be produced without causing any of the sequencers to halt because one or more of the required component types are not already mounted on that sequencer's dispensing heads.

2.2 Problem Statement

The problem facing a production supervisor then, within the planning horizon, is to decide which pack types should be assigned to each available sequencer. This assignment ultimately determines which sequenced tapes will be produced by each sequencer. Recall that management prefers to streamline the production process of the sequenced tapes by reducing the change-over time, or number of change-overs, between consecutive runs on the sequencers. Intruding on this already complex situation is the requirement that the assignment of pack types to sequencers should be such that the distribution of work, or equivalently, total processing time, among the available sequencers is relatively even. Total processing time of a sequencer is measured by the total number of component insertions that the sequencer makes while it produces all of its assigned sequenced tapes within the planning horizon. Management is not easily pleased!

In other words, nirvana, from management's point of view, is characterized by a planning horizon such that once the available sequencers are started up, they are not turned off until all of the sequenced tapes required to be produced are completed <u>and</u> the workload (the number of component insertions that each sequencer makes to produce the tapes) is relatively even. It is easy to envision how the sequencer loading situation can be a production supervisor's nightmare!

Falling short of nirvana, the production supervisor is tasked to devise a pack type-to-sequencer assignment plan that necessitates the fewest number of change-overs between sequencer runs. Also, if change-overs should occur, they should take as little time as possible. All the while, the relatively even distribution of sequencer workload should be maintained. A more detailed explanation of change-overs and how they impact upon the problem solution is given in Chapter 3. They are not an area for consideration in the model developed by Fathi and Taheri since their model totally eliminates all change-overs.

To further amplify the situation where a change-over does not occur, let us consider just one sequencer of those available in a typical problem environment. Assume that this particular sequencer is scheduled to produce 10 sequenced tapes. For this sequencer not to experience a change-over while producing the 10 sequenced tapes, the total number of different component types required to produce the 10 sequenced tapes may not exceed 100, which is the number of dispensing heads on that sequencer. In other words, the cardinality of the union of all of the component types required to produce the 10 sequenced tapes may not exceed 100. If it did, then at some point in time, at least once, the sequencer would have to halt so that the required unmounted component tape(s) could be loaded on a dispensing head(s) of the sequencer, in turn causing other component tape(s) to be removed. The production process of the sequenced tapes would resume after the component tape changes were completed.

2.3 Model Strategy

The strategy employed by Fathi and Taheri in the development of their model is aimed at the complete elimination of all change-over time. They succeed in accomplishing this goal, which is also management's ideal goal, by permanently dedicating specific sequencer dispensing heads to specific component types. In this manner, for every sequenced tape that must be produced, there is at least one sequencer which has all of the necessary dedicated heads. The caveat attached to their model, however, is that depending on the set of input parameters constituting any particular problem, the resulting solution might require the acquisition of additional dispensing heads, which management may or may not decide to purchase.

Prior to further explanation of their model, detailed clarification as to the difference between the sequencer assignment problem and the sequencer scheduling/assignment problem is appropriate.

In the event that management approves of purchasing additional dispensing heads, the Fathi/Taheri model of a sequencer assignment problem is perfectly suitable to determine a pack type-to-sequencer assignment plan. There would never be cause for concern when their model provided a resulting solution requiring additional dispensing heads to be purchased. Under these circumstances, a schedule that exactly orders the production of sequenced tapes for the purposes of reducing change-over time is not required. The Fathi/Taheri model solution dictates that when a pack type is assigned to a sequencer for production, a dispensing head on that sequencer has been dedicated for every one of the different component types required to produce that corresponding sequenced tape. Therefore, the exact order by which different sequenced tapes are successively produced on each sequencer is inconsequential. None of the sequencers will necessitate a change-over regardless of the order of production of successive sequenced tapes.

This property is a result of the construction of their model. The model seeks a particular dispensing head-to-component type dedication, and the corresponding pack type-to-sequencer assignment plan, such that no change-overs are required throughout the planning horizon (except, of course, for when a component tape exhausts its components). In other words, the Fathi/Taheri model seeks to <u>assign</u> pack types to the sequencers in a manner such that all of the different component types required to produce each corresponding sequenced tape are mounted on at least one of the available sequencers. The model makes no attempt to <u>schedule</u> the order of production of sequenced tapes, since, as it has been previously pointed out, a production schedule has absolutely no effect on the degree of goodness of the solution.

By association, then, the Fathi/Taheri model is suitable for use in situations where the assignment of specific pack types to sequencers for production is the primary objective of the analysis, and production scheduling has no impact. The circumstances that propagate this problem (the sequencer assignment problem) are, for any set of typical problem parameters, either the case where the model solution specifying the pack type-to-sequencer assignment plan does not require any additional dispensing heads on any of the available sequencers, or, if the resulting solution does indeed require extra heads to be added, management is willing to purchase them.

Details of the sequencer scheduling/assignment problem are fully discussed in Chapter 3. It is sufficient at this point to note that the sequencer scheduling/assignment problem presents itself whenever change-overs must occur during the production process of the sequenced tapes. If change-overs are necessary during production, the pack type-to-sequencer assignment plan <u>and</u> the exact order of production of sequenced tapes impact upon the number and duration of change-overs. Intuitively, if the pack types assigned to a sequencer are relatively similar, and furthermore, if the order of production of the corresponding sequenced tapes is specified such that the different types of components required to produce each one successively are relatively similar, it is justifiable to expect that the number and the duration of change-overs would be less than the case where the pack type-to-sequencer assignment plan and the corresponding schedule (the order of production of different sequenced tapes on a sequencer) is determined hapzardly.

Therefore, not only does judicious assignment of the pack types to sequencers affect the degree of goodness of the solution, but also, the schedule designating the exact order of production of the assigned sequenced tapes on each sequencer has a significant impact upon the goodness of the solution. This more realistic sequencer scheduling/assignment problem is a product of management's decision not to purchase additional dispensing heads, but instead, to tolerate a certain amount of change-over time.

Returning to the Fathi/Taheri mathematical programming model, the strategy employed in developing their model has been previously discussed. To facilitate further explanation, however, that strategy is reiterated. The model seeks a particular dispensing head-to-component type dedication, and the corresponding pack type-to-sequencer assignment plan, such that no change-overs are required over the planning horizon and the total workload is relatively balanced among the available sequencers during that period. Achieving such a dedication/assignment plan may not be feasible, however, within the current availability of dispensing heads on the sequencers. Therefore, a resulting solution, given any set of typical problem parameters, may require addition of extra heads to the sequencers. The objective function of their model is designed to obtain a dedication/assignment plan requiring the fewest additional heads.

To accomplish this, two different sets of decision variables are used in the model. The first set of decision variables govern the dedication of dispensing heads to component types, while the second set govern the assignment of pack types to sequencers. Some constants describing the sequencing environment and a detailed description of the two sets of decision variables follow:

L - The number of available sequencers.

M - The total number of pack types to be produced.

N - The total number of different component types.

C_k - The number of dispensing heads currently available on the k^{th} sequencer, for k = 1 to L.

Recall that a typical problem environment consists of 3 to 6 available sequencers, 20 to 40 pack types to be produced, and anywhere from 200 to 400 different component types which are needed for production of the required pack types.

The decision variables used in the model are:

x_{jk} - This is a 0-1 variable (for j = 1 to N and k = 1 to L) indicating the allocation of the different component types to the available sequencers. $x_{jk} = 1$ if component type j is allocated to the k^{th} sequencer and $x_{jk} = 0$ otherwise. The number of 'x' decision variables in the model is determined by the product of the number of available sequencers, L, and the number of different component types, N, in any typical problem. In other words, there are N*L 'x' decision variables in any given problem. Notice that if $x_{jk} = 1$, then one (and only one) dispensing head on the k^{th} sequencer will be <u>dedicated</u> to component j.

y_{ik} - This is a 0-1 variable (for i = 1 to M and k = 1 to L) specifying the <u>assignment</u> of the pack types to the available sequencers. $y_{ik} = 1$ if pack type i is assigned to the k^{th} sequencer (that is, the sequenced tape corresponding to pack type i will be prepared by the k^{th} sequencer) and y_{ik} = 0 otherwise. The number of 'y' decision variables in the model is determined by the product of the number of available sequencers, L, and the number of pack types to be produced, M, in any typical problem. In other words, there are M*L 'y' decision variables in any given problem.

The Fathi/Taheri model is categorized as a pure 0-1 integer programming (IP) model, as all of the model decision variables (x_{jk}'s and y_{ik}'s) may only take on the values of 0 or 1. Furthermore, the IP model contains L(M+N) decision variables. That is, since the number of 'x' decision variables is N*L, and the number of 'y' decision variables is M*L, then the total number of decision variables, given any set of typical problem parameters, is the sum of the two, or L(M+N). Relating this formula to the figures describing a typical problem environment (L = 3 to 6 sequencers; M = 20 to 40 pack types; N = 200 to 400 different component types), their IP model could have anywhere from 660 to 2,640 decision variables. The IP model for the sequencer assignment problem is presented in Fathi and Taheri [1986, p. 12].

In addition to the objective function, their model consists of four distinct groups of constraints, of which only one will be addressed in depth. Recall that the two goals of management are minimization of change-over time between consecutive runs on the sequencers, and a relatively even distribution of work among the available sequencers throughout the planning horizon. To achieve the second goal, a group of load-balancing constraints is introduced into the model. These load-balancing constraints are designed to limit the total workload assigned to each sequencer during the planning horizon. As mentioned earlier, workload is measured by the total number of component insertions that a sequencer makes while it produces all of its assigned sequenced tapes. It seems reasonable to assume that the total workload assigned to a sequencer is proportional to the total number of individual components inserted by that sequencer. As a preface to the derivation of these load-balancing constraints, additional notation must be defined:

A - An M by N matrix where (for $i = 1$ to M and $j = 1$ to N) $a_{ij} = 1$ if pack type i requires component type j, $a_{ij} = 0$ otherwise.

b_i - The number of different component types on pack type i, for $i = 1$ to M. Notice that

$$b_i = \sum_{j=1}^{N} a_{ij} \qquad \text{for} \qquad i = 1 \text{ to } M$$

b_i' - The total number of components on pack type i, for $i = 1$ to M. Notice that $b_i' \geq b_i$, due to the fact that pack type i might require more than one unit of any particular type of component.

v_i - Production volume for pack type i, that is, the total number of units of pack type i to be produced for $i = 1$ to M over the planning horizon. Notice that sequencing is a merging operation, thus $b_i'v_i$ is a measure of the total amount of work performed by a sequencer to produce the total demand for the pack type i ($b_i'v_i$ is the total number of components to be sequenced for all packs of type i).

For k = 1 to L, let S_k be the maximum number of components permitted to be inserted by the k^{th} sequencer during the production period. The k^{th} constraint in this group will then be

$$\sum_{i=1}^{M} v_i b_i' y_{ik} \leq S_k \qquad \text{for} \qquad k = 1 \text{ to } L$$

The values for all S_k's must be provided in advance via managerial policy. If management's policy is the relative even distribution of the total workload among all of the available sequencers, then S_k (for k = 1 to L) could be the total number of components to be mounted during the production period divided by L, the number of available sequencers, plus a relatively small constant. That is, let

$$S_k = \frac{\sum_{i=1}^{M} v_i b_i'}{L} + \text{constant} \tag{1}$$

On the contrary, if management's policy specifies an uneven distribution of the load for whatever reason, then the values of the S_k's would need to be fixed accordingly. In any case, it is important that the selection of the values of the S_k's is such that the sequencer assignment problem remains feasible.

2.4 Model Performance

A variety of procedures are suitable for solving the Fathi/Taheri mathematical model. The LINDO computer program [1984] is a system which solves linear and 0-1 integer programming models. The solution procedure used by LINDO to solve integer programs is based around the enumerative method known as branch-and-bound. For a detailed discussion of integer program solution methods, see Salkin [1975] and Ozan [1986].

To provide a solution to an integer program (IP), LINDO first solves the model using linear programming (LP) techniques. See Murty [1983]. The ordinary LP solution generally assigns fractional values to some or all of its decision variables. For a minimization problem, the LP solution provides a lower bound on the optimal solution to the IP. By chance, if the LP solution is all integer, then the LP solution is, in fact, an optimal solution for the IP. Barring this circumstance, LINDO fixes the user-designated integer variables to either 0 or 1 through branch-and-bound procedures to derive an integer solution. In order to utilize LINDO, the IP model must be specifically formulated into an objective function and a body of constraints. This model formulation is based on input parameters and assembled by a matrix generator.

The size and structure of the model formulation depends on input parameters and known data associated with the input parameters. The input parameters are:

L - The number of available sequencers.

M - The total number of pack types to be produced.

N - The total number of different component types.

For any typical combination of these three parameters, the model formulation consists of the objective function and M + 2L + (L*M) constraints. The total number of decision variables (x_{jk}'s and y_{ik}'s) in the model formulation equals L(M+N). The total number of 'x' integer variables equals L*N. The total number of 'y' integer variables equals L*M.

Other known data required to completely define the model formulation are: the component requirements, by number and type, for each pack type to be produced; the number of units of each pack type to be prduced (v_i); the number of available heads per sequencer (C_k); and a volume capacity (S_k), measured in terms of component insertions, representing the maximum workload that may be assigned to each sequencer. Throughout this thesis, the sequencer volume capacities are computed according to equation (1), so that a relatively even workload distribution is maintained. Any deviation from this even workload distribution concept is explicitly noted.

When the model formulation is completely defined by the input, it is then assembled by a matrix generator. The purpose of the matrix generator is to fashion the model formulation into a specific format recognizable to LINDO. The matrix generator is programmed as a subroutine that LINDO calls as it solves the problem. (See Appendix 8.1) For a detailed discussion of matrix generators, see the LINDO references [1984, 1986].

To initiate the test phase of the Fathi/Taheri IP model, an extremely atypical set of input parameters (L = 3 sequencers; M = 3 pack types; N = 4 different component types) and associated known data were generated. The test problems were solved by LINDO on a VAX 11/750 computer. The user must designate those variables that are to be integer-valued. Otherwise, LINDO may assign fractional values to those variables in its final solution. This particular problem has 3(3+4) = 21 total integer variables. There are 3*4 'x' integer variables, and 3*3 'y' integer variables. These input parameters describe an incredibly small problem, but the problem serves to reveal a very interesting phenomenon.

The initial LP solution assigned some integer values, and some fractional values, to the 9 'y' variables as expected. However, integer values of 0 or 1 were assigned to all 12 'x' variables. To satisfy our curiosity, numerous other small problems were solved by LINDO. Each time in the initial LP solution, values of 0 or 1 (rather than any fractional values) were assigned to all 'x' variables, even though LINDO was never explicitly instructed to do so. This phenomenon could be due to the special structure of the IP model, although, as of yet, we have not been able to fully explain it. It was hypothesized that this phenomenon would have a significant favorable impact on the overall computational requirements of the branch-and-bound enumerative procedure used by LINDO.

The logic supporting this hypothesis centered on the fact that any given problem has L(M+N) decision variables which must be integer-valued. If all of the 'x' decision variables are always assigned values of 0 or 1 in any optimal solution to the corresponding LP model, then only the 'y' decision variables would have to be identified to LINDO as integer variables. This results in a reduction from L(M+N) to L*M decision variables that must be designated as integer variables. Recalling typical sequencing environment values for L, M, and N, this results in almost 91% fewer integer variables that would constitute the branch-and-bound node network. Simply, if the branch-and-bound network is significantly reduced in size, it is reasonable to assume that the search algorithm utilized by LINDO would be less time-consuming, thus leading to an optimal solution in a shorter time span.

The matrix generator that formulates the sequencer assignment problem for LINDO was altered to take advantage of this phenomenon. The same problem was entered again, and an optimal IP solution was achieved very quickly. A second, larger problem was generated for testing, although still much smaller than a typical problem. The input parameters for this larger problem are:

L = 3 sequencers; M = 10 pack types; N = 20 different component types.

The number of dispensing heads per sequencer (C_k) = 10, and the volume capacity per sequencer (S_k) remained relatively balanced.

The 3*10 'y' decision variables were identified to LINDO as integer variables, and some 'y' variables were assigned fractional values in the LP solution as expected. It was discovered during the succeeding branch-and-bound phase, in which LINDO assigns 0-1 values to all of the 'y' variables, that the model required an extraordinarily long computation time to achieve an optimal IP solution. The computer required a CPU time of roughly 46 minutes, and performed 99,443 pivots to achieve an optimal IP solution. The branch-and-bound network consisted of 3,689 branches. The reader should note that this problem has a total of 3(10+20) = 90 decision variables. The smallest problem in a typical sequencing environment has 3(20+200) = 660 decision variables. This sample problem is 86% smaller than the smallest typical sequencer assignment problem. One can only imagine how much time the computer would need to solve a typical problem using branch-and-bound techniques.

An attempt was made to unconstrain this sample problem to an atypical extent to determine if the model was sensitive to certain parameters. The volume capacity (S_k) of each sequencer was computed such that all of the pack types could be assigned to only one of the available sequencers. In effect, the workload distribution was skewed so that any sequencer would be permitted to produce all of the sequenced tapes, thus causing an imbalance in the workload distribution. Additionally, the number of heads per sequencer was increased to 17. This is an inordinately high number in comparison to the total number of different component types (20).

To achieve an optimal IP solution for this less constrained sequencer assignment problem, the computer required a CPU time of roughly 14.5 minutes, and performed 29,371 pivots. The branch-and-bound network consisted of 1,571 branches.

It is apparent that this variation of a sequenced tape production scheduling problem is not readily solved to optimality through branch-and-bound procedures. Recall that optimality for the sequencer assignment problem equates to total elimination of all change-over time between consecutive runs on the sequencers. Previous research indicates that large-scale zero-one linear programming problems, such as the sequencer assignment problem, have chronically recorded unreasonable computation times to achieve optimal solutions. An article by Crowder, Johnson and Padberg in the Operations Research journal [1983] contains a study of ten large-scale zero-one linear programming problems. To paraphrase, their report strongly confirms that a combination of problem preprocessing, cutting planes, and clever branch-and-bound techniques permits the optimization of sparse large-scale zero-one linear programming problems. They state that even after problem preprocessing, the remaining gap between the optimal LP solution and the optimal IP solution is "still too large to permit one to expect completion of the branch-and-bound phase within a reasonable time limit" [1983, p. 829].

Spielberg [1979, p. 157] writes

> . . . large scale LP problems can still be inordinately difficult on account of degeneracy, and there seems no prospect for easy remedies. The node problems of the BB (branch-and-bound) approach may therefore prove to be too difficult, and enumerative approaches (or heuristic ones; which are intrinsically more related to enumeration than to BB programming) may be required.

Obviously, Spielberg is leaning towards alternate approaches to solve large-scale zero-one linear programming problems. The subject matter of the following chapters describes a heuristic approach to another variation of the sequenced tape production scheduling problem.

The sequencer scheduling/assignment problem is created by management's decision not to purchase extra dispensing heads, but instead, to tolerate some sequencer change-over time. Due to this self-imposed constraint, it is very doubtful that this problem can be solved to optimality in polynomial time. McGinnis et al. [1986] report that this sequencer scheduling/assignment problem is NP-hard. Therefore, one practical approach to this problem is a heuristic procedure which is computationally efficient, although it may result in a suboptimal solution.

Papadimitriou and Steiglitz [1982] explain the definition of NP-hard problems, and discuss the concept by which a problem is classified as NP-hard. Additionally, they offer some insight into the value of heuristic approaches. They describe a heuristic as any approach without a formal guarantee of performance, and assert that such approaches are certainly valid in practical situations. Development of such a heuristic approach is the topic of the remaining chapters of this thesis.

CHAPTER 3

THE SEQUENCER SCHEDULING/ASSIGNMENT PROBLEM

3.1 Problem Background

The sequencer scheduling/assignment problem portrays a realistic situation involving constrained resources (sequencer dispensing heads). Consider, again, a situation where a certain number of sequenced tapes, M, must be produced within the planning horizon (typically, $20 \leq M \leq 40$). The problem surfaces when the total number of different component types required to produce all of the sequenced tapes exceeds the total number of dispensing heads on the available sequencers. Therefore, it is physically impossible to load all of the required component tapes on the sequencers at one time. This situation demands at some point in time during the planning horizon that at least one of the sequencers will have to halt for unloading/loading of component tapes in order to produce all of the sequenced tapes.

Even if the total number of different component types does not exceed the total number of available dispensing heads, it might still not be possible to allocate different component types to dispensing heads of different sequencers so that all of the different component types required for each sequenced tape are available on one sequencer. In any case, the thrust of the remaining chapters of this thesis is focused on the scheduling situation involving imminent sequencer change-overs. Also, the reader should construe 'component types' to mean 'different component types.' The word 'different' is frequently omitted from this point forward.

Mindful of management's desire to streamline production operations, a concerned production supervisor, in light of this impending situation, would immediately approach management with a request to purchase extra dispensing heads. "We can eliminate all change-over time by adding extra heads." The Fathi/Taheri model can be used to determine the fewest number of additional heads to purchase, he could argue. Faced with management's very final decision not to purchase extra heads, but instead, to tolerate some (minimal) change-over time, the dejected production supervisor realizes that he is confronted with a very complex, formidable task.

3.2 Problem Statement

By some method, a schedule that assigns the sequenced tapes to be produced to the available sequencers must be devised. In addition, this production schedule must designate the exact order by which different sequenced tapes are produced on each sequencer. Obviously, this production schedule dictates which component tapes are loaded onto the individual dispensing heads, and when they are to be loaded. An efficient production schedule serves to reduce the inevitable change-over time between consecutive runs on the sequencer(s).

As mentioned earlier, development of an optimal schedule could be, computationally, quite difficult. Our objective, then, is to develop a heuristic procedure that devises such a production schedule in a reasonable amount of time, even though the resulting schedule may not be an optimal one. This heuristic procedure would be aimed at reducing the total change-over time between consecutive runs on the sequencers for a given set of pack types and their associated volume requirements.

In addition, the heuristic procedure should ensure a relatively even distribution of total workload among the available sequencers throughout the sequenced tape production process.

3.3 Strategy of Heuristic Approach

In order to develop a heuristic procedure that provides a solution to the sequencer scheduling/assignment problem, the objective of the procedure must be clearly defined. The formal objective is previously stated in Section 3.2. Informally stated, the objective is to allocate the requisite component types to the available dispensing heads such that the total sequencer change-over time is small and that each sequencer roughly makes a similar number of component insertions over the planning horizon. The sequenced tapes to be produced compete for dispensing heads. Some sequenced tapes have similar component type requirements, while others may be vastly different. In order to capitalize on the number of existing dispensing heads, it is logical to assign sequenced tapes that require similar component types to the same sequencer.

The heuristic approach pursued in order to provide a solution to the sequencer scheduling/assignment problem embodies this logical thought process. The fundamental strategy employed to confront the sequencer scheduling/assignment problem is to associate together those pack types which require similar component types. Conversely, those pack types that bear little resemblance to one another are disassociated by assigning them to different sequencers when possible (recall the relatively balanced workload restriction).

This strategy governs not only the assignment of pack types to sequencers, but it also governs the order of production of the sequenced tapes on each sequencer. When the sequenced tapes have been assigned to a sequencer, the exact order by which they are successively produced is necessary. Ordering their successive production based on the similarity of their respective component type requirements would likely result in shorter, and possibly fewer sequencer change-overs.

3.4 Measures of Goodness

Three different heuristic procedures are developed to provide solutions to the sequencer scheduling/assignment problem. The inherent structure of each of these procedures is founded upon the fundamental strategy outlined in Section 3.3. When different heuristic procedures are developed to *provide solutions to the same problem*, various measures of goodness may be identified that permit comparison of the different procedures with respect to their individual performance. In an empirical sense, conjectures as to their relative merit may be made after numerous typical problems are solved.

In this section, the primary and secondary measures of goodness by which the three different heuristic procedures are compared are thoroughly discussed. These two measures of goodness are valid discriminators which accurately gauge the overall utility of the resulting production schedules. Other, less significant measures of goodness are considered. These less significant measures are marginally useful when the performance of the three different heuristic procedures is extremely competitive with respect to the primary and secondary measures of goodness. In other words, when production schedules devised by the three different procedures, for a particular set of problem parameters, cannot be conclusively gauged by the primary and secondary measures of goodness, these less significant measures of goodness may provide some insight as to their relative performance.

The primary measure of goodness for the three different heuristic procedures is the total number of change-overs that occur in any production schedule. The major contributor to an extended sequencing planning horizon is sequencer halts. A production schedule that causes two sequencer change-over periods is categorically worse than a sequenced tape production schedule that causes only one change-over period. (A sequencer that makes two change-overs is exactly equivalent to two sequencers making one change-over apiece.) Sequencer change-over periods are prominent in this respect because the set-up operations required to prepare a sequencer for processing consume the most time, and adversely extend the planning horizon more than any other operation. Given a choice of production schedules devised for the same set of problem parameters, the production supervisor would always select the schedule requiring the fewest number of change-over periods. Further explanation of the high level of prominence placed on the number of sequencer change-overs is in order.

Ordinarily, a production supervisor might have reason to debate the overall utility of different production schedules devised for the same set of problem parameters. Recall that management has two goals for production. The first goal is minimal sequencer change-over time, and the second goal is a relatively even workload distribution. The production supervisor would be in a quandary, for example, if one production schedule required only one sequencer change-over, but had an unbalanced workload distribution, while another production schedule required two change-over periods, but maintained a relatively balanced workload distribution. With respect to management's two goals, selection of the better production schedule is possibly ambiguous.

Fortunately, the three different heuristic procedures are structured not to permit this set of circumstances. The three procedures maintain a relatively balanced workload at all times, thereby always satisfying the second of management's two goals. For this reason, the total number of sequencer change-overs in any production schedule is the paramount discriminator in gauging their relative performance.

The secondary measure of goodness is the total number of component tape changes necessitated by a given production schedule. This secondary measure is especially relevant when the resulting production schedules devised for the same problem all specify an identical number of sequencer change-overs. When a particular sequenced tape is designated for production, and all of the component types required for that sequenced tape are not currently mounted, the sequencer must halt. Component tapes not needed for that sequenced tape are removed to make dispensing heads available for the required unmounted component tapes. Intuition concedes that the total time needed to unload and load component tapes is directly proportional to the number of component tapes that must undergo this operation. Therefore, the second criterion to compare the performance of the three different heuristic procedures is the fewest number of component tape changes required by a production schedule for the same set of problem parameters.

The relative merits of the three different heuristic procedures developed in this thesis are empirically determined strictly with respect to the primary and secondary measures of goodness. Three additional, less significant measures of goodness provide some insight into the comparative performance of the different heuristic procedures. These, for lack of a better term, tertiary measures are: sequencer workload distribution; the total number of different component types required by any sequencer to produce all of its assigned sequenced tapes; and the total number of dedicated dispensing heads per sequencer within a planning horizon.

Production schedule data pertaining to these tertiary measures of goodness is presented. However, that data should be digested only as a marginal consideration with respect to relative performance, as the three different heuristic procedures are not rank ordered based on these tertiary measures. The three tertiary measures merely assist in assessing the performance of the different heuristic procedures when their resulting production schedules devised for the same problem cannot be conclusively gauged by the primary and secondary measures of goodness.

The first tertiary measure of goodness discussed is the sequencer workload distribution. The treatment of the distribution of total sequencer workload over the planning horizon is addressed earlier in this section. The basic structure of the three different heuristic procedures is such that the distribution of total workload is relatively balanced at all times. However, the three procedures may devise entirely different production schedules for the same set of problem parameters. Therefore, the distribution of total workload may be more or less balanced for each production schedule. If the situation demands (the total number of sequencer change-overs, and the total number of component tape changes specified by the resulting schedules of all three procedures are identical for the same problem), the overall sequencer workload distributions generated by the three schedules may be compared to determine which heuristic procedure has the most evenly balanced workload distribution, thus indicating the better of the three schedules.

The second tertiary measure of goodness is the total number of different component types required by any sequencer to produce all of its assigned sequenced tapes. Once all of the sequenced tapes are assigned to the available sequencers for production, the total number of different component types required to produce all of the assigned sequenced tapes on each sequencer is easily calculated. When the required number of different component types exceeds the fixed number of dispensing heads, it is clearly inevitable that that particular sequencer will experience a change-over during the planning horizon.

The duration of any sequencer change-over period is, in part, a function of the number of component tapes changed in that change-over period. Intuitively, the total amount of change-over time for a given sequencer is expected to be decreased as the number of different component types, over and above the number of dispensing heads is also decreased. In other words, if the total number of component tapes that must be changed on a sequencer within a planning horizon is small, it is reasonable to assume that that sequencer will experience a lesser amount of change-over time. The possibility that a component tape may be dismounted, then remounted at a later time in that same production period, is not ruled out. However, an efficient production schedule should make this particular event remote at best.

The final tertiary measure of goodness is the total number of dedicated dispensing heads per sequencer within a planning horizon. The definition of 'dedicated,' in this context, is that once a component tape is mounted onto a sequencer dispensing head, it is never removed (except, of course, for when a component tape exhausts its components). 'Dedicated' dispensing heads surface when the particular component type on that head is required for production by <u>every</u> sequenced tape assigned to that sequencer. The logic supporting this measure of goodness lies in the fact that there are a fixed number of dispensing heads on any sequencer. If the number of dedicated dispensing heads on a particular sequencer approaches its fixed number of heads, by complementarity, there are fewer dispensing heads on that sequencer eligible for unloading and loading of component tapes. A correlation might exist between a large number of dedicated dispensing heads and a reduced amount of change-over time for any given sequencer.

To emphasize, these tertiary measures of goodness are not, in themselves, empirical evidence enough to make a valid distinction between the three different heuristic procedures. They are marginally useful in possibly selecting one heuristic procedure over another when their resulting production schedules are very similar. They are also very subjective in nature, as their results are given to interpretation and production experience.

CHAPTER 4
DESIGN OF HEURISTIC APPROACH

4.1 Developmental Assumptions

For a problem as broad as the sequencer scheduling/assignment problem, some underlying assumptions are relevant in developing a heuristic approach. Through legitimate assumptions, the problem at hand is analyzed in greater depth. This analysis is essential to achieve a satisfactory solution. Two main assumptions are incorporated into the design of this heuristic approach. A third assumption is not incorporated, as information necessary for its incorporation could not be obtained. However, this assumption is discussed because it is associated with the second of management's two production goals--relatively balanced sequencer workload distribution.

The first assumption incorporated into this heuristic approach is that a pack type does not require more component types than there are dispensing heads on any available sequencer. This assumption is previously mentioned in Section 2.1. In a typical sequencing environment, each available sequencer is equipped with a fixed number of dispensing heads. When the number of component types required by an assigned pack type exceeds this fixed number, that particular sequencer requires a change-over just to produce that one corresponding sequenced tape. This situation is highly irregular and would probably lead management to purchase additional heads to produce these outsized sequenced tapes.

The second assumption incorporated into this heuristic approach is that the pack type volume requirements (v_i - the number of each pack type to be produced) is roughly similar. This assumption relates to the definition of sequencer workload and greatly affects the utility of the formula used to compute sequencer volume capacity (S_k). In the sequencing environment, work is measured by the total number of component insertions that a sequencer makes over its planning horizon. Suppose there exists a production volume requirement for a particular pack type that is abnormally higher than all of the other pack type production volume requirements. Two different situations may occur given this particular event.

The first case surfaces when the total number of component insertions required to produce the sequenced tapes for all units of that particular pack type does not exceed the volume capacity (S_k) of any available sequencer. In this case, one sequencer is capable of producing all of those particular sequenced tapes. However, due to the extremely large component insertion requirement attached to that particular sequenced tape, very few different pack types may be assigned to that sequencer. To compensate for this unusual assignment, the remaining sequencers are required to produce a disproportionately large number of different sequenced tapes. It is reasonable to assume that a sequencer stands a better chance of experiencing a change-over as the number of different sequenced tapes it is assigned to produce increases. The sequencer producing the sequenced tape with the abnormally high volume requirement would experience few, if any, change-overs, since it is basically producing one type of sequenced tape.

The second case surfaces when the total number of component insertions required to produce all of the sequenced tapes for that particular pack type exceeds the volume capacity of any available sequencer. In this case, one sequencer is not capable of producing all of those particular sequenced tapes because the computed volume capacity (S_k) is not large enough. Therefore, the assignment of those particular sequenced tapes would have to be divided over multiple sequencers. Both cases mentioned above would require special consideration to be properly treated in a heuristic approach.

A third assumption, not incorporated into this heuristic approach, would be valuable. Sequencer processing time is measured in terms of component insertions. When a sequencer experiences a change-over, it is not inserting components; hence, its processing time is temporarily halted. However, chronological time is still elapsing. A method to relate sequencer change-over time (chronological time) to sequencer processing time (number of component insertions) would be beneficial. As of now, all sequencer change-over periods are considered to be equivalent, regardless of the number of component tapes that are changed during the change-over period. This relation would permit comparison of the relative duration of sequencer change-over periods, thus providing the means by which production schedules with an identical number of sequencer change-overs may be distinguished.

The total chronological time consumed during a sequencer change-over is, in part, a function of the number of component tape changes required. Therefore, it is very likely that different production schedules devised for the same problem, each specifying an identical number of sequencer change-overs, are not completed in the same chronological time. A formula that could be used to relate the duration of a particular sequencer change-over period to equivalent sequencer processing time, in terms of a specific number of component insertions, is.

$$T = K + cX$$

Definitions of the terms in this formula are as follows:

T - The total processing time, in terms of the specific number of component insertions, of one sequencer change-over period.

K - The set-up time incurred during a sequencer change-over period, in terms of a fixed number of component insertions. This is a constant.

c - The time required to change one component tape, in terms of a fixed number of component insertions. This is also a constant.

X - The total number of component tapes that must be changed in one particular sequencer change-over period.

Valid estimates for the values of the constants, K and c, could not be obtained. The routine sequencer set-up time, and the time required to change one component tape, can be roughly equated to the time it takes a sequencer to insert a certain number of components. These constants would most likely be estimated through actual production experience.

This formula equates the chronological time consumed during a particular sequencer change-over period to a specific number of component insertions that a sequencer routinely makes in the same amount of chronological time. By incorporating this formula, the total production volume requirement of any sequencer would accurately reflect the total number of component insertions required to produce its assigned sequenced tapes, plus an equivalent number of component insertions incurred by the change-overs experienced by that sequencer. The algorithm would take these equivalent sequencer production volume requirements into account and balance them accordingly as it devised an appropriate production schedule. In this manner, the <u>total</u> sequencer processing time required by a production schedule would be more accurately gauged. This permits a better insight when comparing the utility of different production schedules devised for the same problem.

4.2 <u>Heuristic Algorithm</u>

The specific algorithm, which assigns all of the pack types to be produced to the L available sequencers, strictly adheres to the strategy discussed in Section 3.3. The algorithm assigns pack types which exhibit a high degree of similarity, in terms of component type requirements, to the same sequencer. The algorithm separates pack types that have little in common, in terms of component type requirements, by assigning them to different sequencers.

In order to assign pack types to the sequencers according to this strategy, the precise meaning of the terms <u>similar</u> and <u>common</u> must be defined for the algorithm. An exact understanding of these two terms enables the algorithm to accurately establish the correct relationship between pack types in terms of component composition.

4.2.1 SIMILAR Pack Types

There exists one precise definition of the term similar. The algorithm searches for similar pack types only when it is in the process of scheduling a sequencer for production. After the algorithm has assigned the first pack type to a sequencer, it selects the most similar, unassigned pack type. The algorithm attempts to assign this most similar pack type to that same sequencer. If that pack type is successfully assigned, the algorithm again selects the most similar, unassigned pack type, and attempts to assign it to that same sequencer. This process is repeated until the algorithm can no longer schedule a sequencer for one of several reasons.

The precise, unwaiverable definition of the term similar, with respect to pack types, in the context of a sequencing environment, is:

> When one or more pack types have been assigned for production to the k^{th} sequencer (for k = 1 to L), the unassigned pack type that would require the fewest number of additional, different component tapes to be mounted on that kth sequencer, so as to produce that pack type together with the previously assigned pack type(s) on the same production run, is, in fact, the most similar, unassigned pack type.

This definition of similar is identical for the three different heuristic procedures. This specific definition of similar capitalizes on the fixed number of dispensing heads, and permits the maximum possible number of sequenced tapes to be produced together on a sequencer, for a particular production run. It is reasonable to assume that a production schedule which always assigns the maximum possible number of sequenced tapes to the kth sequencer for all production runs will, most likely, favorably result in a small amount of total sequencer change-over time.

To emphasize, the algorithm does <u>not</u> select the <u>first</u> pack type assigned to the kth sequencer (for k = 1 to L), for all production runs, by incorporating the definition of the term <u>similar</u>. The algorithm does, however, select <u>all</u> pack types <u>subsequently</u> assigned to the kth sequencer, for all production runs, by solely incorporating the definition of <u>similar</u>.

With one exception, the pack type assigned to the kth sequencer (for k = 1 to L), for all production runs, is selected by incorporating one of three definitions of the term <u>common</u>. The one exception pertains to the very first assigned pack type when the algorithm commences the scheduling process. This first pack type is selected via a specific starting rule that is in no way associated to the definition of the term <u>common</u>. Discussion of this starting rule is presented in Section 4.3.1. The multiple definitions of <u>common</u>, and the method by which the three definitions are implemented in the heuristic approach for the sequencer scheduling/assignment problem, comprise the subject matter of Chapter 5. It is sufficient to note that the three definitions of the term <u>common</u> correspond to the three different heuristic procedures. It is not necessary to know the three definitions of <u>common</u> in order to follow the algorithm as it schedules sequenced tapes for production.

In order for the algorithm to assign subsequent pack types to sequencers based on the definition of the term <u>similar</u>, the actual relationship indicating the degree of similarity between pack types must be determined. The method by which the similar relationship between pack types is established, in terms of component type requirements, is described in the next section.

4.2.2 Degree of Similarity Between Pack Types

Several operations must be performed prior to initiation of the algorithm. The input parameters, L, M, N, and associated known data (including the number, and type, of each component required by each pack type) are entered. With this data, the various component type totals (b_i, b_i', $v_i b_i'$; see pg. 22 for definitions) are computed.

The similar relationship between pack types is established next. The method by which this similar relationship is established is through the formation of a two-dimensional array, named Array DIFFER. The numerical entries contained in DIFFER indicate the degree of similarity between pack types in terms of component type requirements.

Array DIFFER is an (M+L)*M matrix. At this stage of the heuristic procedure, only the first M rows of the total M+L rows are filled with numerical entries. The M+k^{th} row of DIFFER, for k = 1 to L (or equivalently, the last L rows), is initially filled with numerical entries after the algorithm assigns two pack types to the k^{th} sequencer. Discussion pertaining to these last L rows of DIFFER is presented in Section 4.3.4.

The discussion in this section pertains only to the first M rows of Array DIFFER. The numerical entries in the individual elements of the i^{th} row (for i = 1 to M) of DIFFER are derived by comparing the different component types required by M-1 pack types against the different component types required by pack type i. Obviously, pack type i is not compared against itself. The operation by which the different component types required by the M-1 pack types are compared against the different component types required by pack type i, to fill the i th row of DIFFER, is best described as a question:

> If the component tapes required to produce pack type i are already mounted on the dispensing heads of a sequencer, how many more different component tapes have to be additionally mounted in order to produce pack types i and q together on the same sequencer? ($i = 1$ to M; $q = 1$ to M; $i \neq q$)

This question is posed for every possible comparison of two pack types. It is essential to note that the implication of comparing pack type q against pack type i is, strictly, that the component tapes of pack type _i_ are _already_ _mounted_, and it is to be determined if pack type q may be produced together with pack type i. Example 4.1 demonstrates the formation of the first M rows of Array DIFFER.

EXAMPLE 4.1

Suppose that a sequencing environment consists of the following:

L = 2 available sequencers.

M = 3 pack types to be produced.

N = 12 total different component types.

C_k = 10 dispensing heads on the k th sequencer.

The total number of different component types (b_i) required by packs 1, 2, and 3 are 7, 6, and 6, respectively. The following 'A' matrix (Table 4.1) represents the component type requirements of the three pack types.

Table 4.1 Component Type Requirements [matrix A]

Component Type	1	2	3	4	5	6	7	8	9	10	11	12	b_i
Pack type 1	1	1	0	0	1	0	1	1	1	0	0	1	7
Pack type 2	0	1	0	1	1	0	0	1	0	1	1	0	6
Pack type 3	0	0	1	0	1	1	1	0	1	0	0	1	6

Recall that if $a_{ij} = 1$, then pack type i requires component type j for production; $a_{ij} = 0$ otherwise.

In forming the first M rows of Array DIFFER, the component type requirements of pack 1 are compared individually against the component type requirements of packs 2 and 3. The component type requirements of pack 2 are compared individually against the component type requirements of packs 1 and 3. The component type requirements of pack 3 are compared in the same fashion to packs 1 and 2. These comparisons are made by posing the question described earlier in this section. The question is posed for every possible comparison of two pack types (e.g., 1 and 2 together; 1 and 3 together; 2 and 1 together; 2 and 3 together; 3 and 1 together; and 3 and 2 together). Since a pack type is not compared against itself, the main diagonal of the M*M portion of DIFFER is represented by hash marks. The numerical entries in the M*M portion of DIFFER for this example are shown in Table 4.2.

Table 4.2 M*M Portion of Array DIFFER

M = number of pack types = 3

(i = 1 to M)

L = number of available sequencers = 2

(k = 1 to L)

	Packs 1	2	3	
Pack 1	-	3	2	
Pack 2	4	-	5	M rows
Pack 3	3	5	-	
				L rows

The operations required to compare pack 2 against pack 1 are described below (refer to Table 4.1). Identical operations are required for all comparisons of two pack types.

Pack 1 requires seven different component types for production--component types 1, 2, 5, 7, 8, 9, and 12. Pack 2 requires six different component types for production--component types 2, 4, 5, 8, 10, and 11. The question is posed: If the component tapes required to produce pack type 1 are already mounted on the dispensing heads of a sequencer, how many more different component tapes have to be additionally mounted in order to produce pack types 1 and 2 together on the same sequencer? The only different component types that pack 2 requires that are not already required by pack 1 are component types 4, 10, and 11. Therefore, if pack 1 component types are already mounted, only three more different component types must be additionally mounted to produce packs 1 and 2 together. Hence, 3 is the entry in row 1, column 2, of Array DIFFER (Table 4.2), which represents the comparison of component type requirements of pack 2 against the component type requirements of pack 1.

Two items specific to the M*M portion of Array DIFFER are briefly addressed. First, there are always M-1 entries in the i^{th} row (for i = 1 to M) of this portion of the array. Second, the M*M portion of DIFFER is <u>not</u> symmetric. If it was symmetric, the size of DIFFER would be reduced, which would lend itself to a more efficient algorithm. The comparison of pack 2 against pack 1 yields a value of 3, while the comparison of pack 1 against pack 2 yields a value of 4 (see Table 4.2). This is not to say that the corresponding entries will <u>always</u> be different. (See the entries in Table 4.2 for the comparison of pack 2 against pack 3, and the comparison of pack 3 against pack 2.)

To summarize this section, a numerical entry in the q^{th} element (for q = 1 to M) of the i^{th} row (for i = 1 to M; $i \neq q$) of Array DIFFER strictly indicates the degree of similarity between pack type q and pack type i. This particular array element represents the comparison of the component type requirements of pack type q <u>against</u> the component type requirements of pack type i. The <u>smallest</u> numerical entry in the i^{th} row represents the pack type that is <u>most</u> <u>similar</u> to pack type i; the <u>largest</u> entry represents the <u>least</u> <u>similar</u>.

Referring to Example 4.1, if pack type 1, for instance, is first assigned to a sequencer, pack type 3 is the most <u>similar</u>, unassigned pack type, with respect to pack type 1, because row 1, column 3 contains the smallest numerical entry (2). This numerical entry of 2 indicates that, given the seven different component tapes required to produce pack type 1 are already mounted ($b_1 = 7$), two different component tapes must be additionally mounted to produce pack types 1 and 2 together on the same sequencer.

Notice that the specific definition of <u>similar</u> capitalizes on the fixed number of dispensing heads in Example 4.1. The number of dispensing heads (C_k) for this example equals 10. Assuming again that pack type 1 is first assigned, pack type 3 is the most <u>similar</u>, unassigned pack type. Producing pack types 1 and 3 together requires a total of nine different component tapes. Producing pack types 1 and 2 together requires a total of $7 + 3 = 10$ different component tapes. By utilizing the definition of <u>similar</u>, the number of dispensing heads required for production is minimized.

After the first M rows of Array DIFFER are formed, the heuristic algorithm is initiated. The algorithm itself is basically composed of two parts. In the first part, the algorithm seeks an initial assignment of pack types to <u>all</u> available sequencers. The computed sequencer volume capacities (S_k's), with a relatively small constant inserted in equation (1), ensure that each available sequencer is assigned <u>at least</u> one pack type in this first production run. In this thesis, a constant of 10% is inserted to gather the empirical data used to compare the three different heuristic procedures. If the constant in equation (1) was fairly large (indicating an unbalanced workload distribution), the possibility exists that sequenced tapes would not be assigned to every available sequencer.

The second part of the algorithm assigns pack types not previously scheduled for production until none remain. This second part selects particular sequencers for multiple production runs based strictly on their individual production volume requirements. The sequencer with the fewest total number of incurred component insertions required to produce all of its previously assigned sequenced tapes is always selected to make the next production run, if one is required. When the second part of the algorithm terminates, all pack types are assigned to one of the available sequencers, and a detailed production schedule is prepared.

4.3 Heuristic Algorithm - Part One

Generally described, the sequenced tape assignment process begins in this first part of the algorithm by scheduling the first of L available sequencers. Similar pack types are consecutively assigned to this first sequencer until one of two possibilities occurs. The first possibility is that the number of component types required by the assigned pack types successively decrements the number of available dispensing heads such that additional pack types may not be assigned for this first production run. A sufficient number of unallocated heads must be available to accommodate the additional, unmounted component tapes required to produce any unassigned pack type. The second possibility is that the total number of component insertions required by the assigned pack types approaches the computed sequencer volume capacity (S_k), and the scheduling of any additional, unassigned pack types would violate the volume capacity of this first sequencer.

When either of these two possibilities manifests itself, the algorithm leaves the first sequencer and assigns pack types to another sequencer by utilizing Array DIFFER. This process is repeated for all L sequencers. When the final Lth sequencer is being scheduled, the two possibilities described above may, indeed, occur. For whichever of these two reasons prevents additional pack types from being assigned to the Lth sequencer, the algorithm proceeds with the second part.

However, depending on the nature of the component type requirements of the pack types to be produced, all of the remaining, unassigned sequenced tapes might be feasibly scheduled on this Lth sequencer. In this case, the algorithm terminates, thus precluding the need for the second part of the algorithm. The devised production schedule would reflect that each sequencer makes one production run only, thereby indicating that no sequencer change-overs are required.

The detailed description of this first part of the heuristic algorithm is decomposed into five distinct segments. The first segment describes the method by which the first pack type is assigned to the L available sequencers. The second segment describes two bookkeeping operations performed by the algorithm after the first pack type is assigned to any of the L sequencers. The third segment describes the method by which the second pack type is assigned to each of the L available sequencers. The fourth segment describes a comparison operation always performed by the algorithm after the second pack type is assigned to any of the L sequencers. The fifth segment describes the method by which remaining, unassigned pack types are repeatedly assigned to each of the L available sequencers until one of the two previously mentioned possibilities occurs, upon which the second part of the heuristic algorithm is necessary, or until all of the pack types are assigned, in which case, the sequenced tape production schedule is completed.

Numerous operations and procedures performed in the first part of this algorithm are also performed in the second part. When these situations occur, they are subtly identified by the inclusion of the phrase 'for all production runs.' This terminology indicates that the operations or procedures presently being described are also applicable in the second part of the algorithm.

4.3.1 Assignment of the First Pack Type to Sequencers

Two separate rules govern the selection of the first pack type assigned to each of the L available sequencers. One rule, the starting rule, governs the selection of the first pack type assigned to the very first available sequencer. The second rule, the 'least common' rule, governs the selection of the first pack type assigned to the remaining L-1 available sequencers. These two rules, and the logic supporting them, are discussed separately.

The first part of this heuristic algorithm is initiated with a specific starting rule. This rule states that the first pack type assigned to the very first available sequencer is that pack type requiring the fewest total number of different component types. This starting rule selects that pack type with the smallest b_i (for $i = 1$ to M). If more than one pack type qualifies for selection according to this starting rule, the tie is broken arbitrarily.

The immediate logic supporting this starting rule relates to the concept that sequenced tapes compete for dispensing heads. Obviously, the assignment of a pack type requiring the smallest number of dispensing heads implies that the largest possible number of unallocated dispensing heads remains. By associating similar pack types together on the same sequencer, this starting rule should assist in capitalizing on the fixed number of available dispensing heads, thus leading to more efficient production schedules.

The selection of the first pack type assigned to the remaining L-1 available sequencers is governed by the 'least common' rule. The algorithm attempts to assign pack types together on the same sequencer based on their high degree of similarity. Conversely then, the algorithm attempts to separate those pack types that have little in common, in terms of component type requirements, by assigning them to different sequencers.

Therefore, when the algorithm begins scheduling the remaining L-1 available sequencers, the first pack type selected is that pack type whose component type requirements are least common with respect to the component type requirements of all of the pack types already assigned on all of the previously scheduled sequencers. The logic supporting this 'least common' rule stems directly from the strategy of separating dissimilar pack types in order to reduce change-over time.

If the first assigned pack type has little in common with all previously assigned pack types, then, logically, all of the similar pack types assigned with this least common pack type should also have little in common with all previously assigned pack types. In this manner, the strategy of the heuristic approach is embellished. The 'least common' rule is discussed in Chapter 5.

4.3.2 Bookkeeping Operations after Assignment of Pack Types

Two bookkeeping operations performed by the algorithm immediately after the first pack type is assigned for production to any of the L available sequencers merit discussion. In reality, these two operations are intrinsic to the algorithm, and are performed not only after assignment of the first pack type, but after the assignment of all pack types. However, there is a deviation in one of the bookkeeping operations depending on whether a particular pack type is assigned first to a sequencer, or assigned thereafter. Therefore, the discussion commences concerning the first assigned pack type. The deviation in the one bookkeeping operation is explained when the situation warranting that deviation surfaces.

When the production period begins, all of the sequencer dispensing heads are unallocated, and the production volume requirement (total number of required component insertions) of each sequencer equals zero. When a pack type is assigned for production to the kth sequencer, the number of dispensing heads allocated to produce that assigned pack type is incremented. The production volume requirement (PVR) of the kth sequencer is incremented to reflect the total number of component insertions required to produce that assigned pack type. Notation used to portray these two operations is as follows:

H_k - This is the number of dispensing heads allocated to produce all pack types assigned to the k^{th} sequencer, for a particular production run. When the k^{th} sequencer undergoes scheduling for a particular production run, H_k (for k = 1 to L) = 0. Every time a pack type is assigned to the k^{th} sequencer, the number of additional component tapes required to be mounted, in order to produce that pack type, is added to H_k. Recall that C_k is the total number of available dispensing heads on the k^{th} sequencer. The algorithm assigns pack types to the k^{th} sequencer until H_k approaches, but does not violate, C_k.

PVR_k - This is the total Production Volume Requirement of the k^{th} sequencer incurred by the assignment of all of its pack types over all production runs. In other words, PVR_k reflects the total number of component insertions that sequencer k must make in order to produce all of its assigned sequenced tapes. When the production period begins, PVR_k = 0 (for k = 1 to L). Every time a pack type (pack type i) is assigned to sequencer k, the total number of component insertions required to produce pack type i (called PVR_i) is added to PVR_k. In this first part of the algorithm, PVR_k may not exceed S_k, the computed volume capacity of the k^{th} sequencer. The sequencer volume capacities (S_k's) are only enforced in this first part. S_k does not restrict the algorithm in the second part. The algorithm assigns pack types to the k^{th} sequencer in this first part until PVR_k approaches, but does not violate, S_k.

Therefore, after the first pack type is assigned to the kth sequencer (for k = 1 to L), the number of allocated dispensing heads (H_k) is incremented from 0 to the total number of different component types (b_i) required to produce that first assigned pack type i.

$$H_{k,current} = b_i \tag{2}$$

The production volume requirement (PVR_k) is incremented from 0 to the production volume requirement (PVR_i) of that first assigned pack type i.

$$PVR_{k,current} = PVR_i \tag{3}$$

The algorithm considers these currently updated, incremented values of H_k and PVR_k before it assigns another pack type to the kth sequencer. By doing so, the algorithm ensures that subsequent scheduling of unassigned pack types does not violate the sequencer dispensing head limits (C_k), nor does it violate the sequencer volume capacity (S_k).

4.3.3 Assignment of the Second Pack Type to Sequencers

After the bookkeeping operations are completed, the algorithm utilizes only one criterion to select the second pack type assigned to all L sequencers. It selects the unassigned pack type that is most similar to the first pack type already assigned on the kth sequencer (for k = 1 to L). The algorithm locates this most similar unassigned pack type by specifically searching the ith row of Array DIFFER, for i = 1 to M (corresponding to the first assigned pack type i). The algorithm selects the smallest numerical entry corresponding to an unassigned pack type (0 is a possible entry).

This smallest numerical entry represents the unassigned pack type that is most similar to the first assigned pack type in terms of component type requirements. The notation and appropriate definitions to identify these numerical entries in the M*M portion of DIFFER are provided below.

b_{iq} - Let b_{ij} be the numerical entry in the j^{th} column of the i^{th} row of Array DIFFER (for $j = 1$ to M; $i = 1$ to M; $j \neq i$). b_{ij} indicates the degree of similarity between pack types i and j in terms of their component type requirements. Let b_{iq} = minimum $\{b_{ij} : j = 1$ to M; $j \neq i$. Pack type q, then, is most similar to pack type i. If b_{iz} = maximum $\{b_{ij}; j = 1$ to M; $j \neq i\}$, then pack type z is least similar to pack type i.

If multiple, unassigned pack types appear to be most similar to pack i (multiple b_{iq}'s in the i^{th} row), some computations are performed to determine which unassigned pack type is actually more similar. Example 4.2 demonstrates this tie-breaking procedure appropriately.

EXAMPLE 4.2

Suppose that three pack types are to be produced by the k^{th} sequencer. Pack 1, pack 2, and pack 3 require 10, 10, and 9 different component types, respectively. Assume that pack 1 is the first assigned pack type, and the algorithm is attempting to select the second, most similar pack type for assignment next (either pack 2 or 3). The first row of Array DIFFER, corresponding to pack type 1, and the totals of the different component type requirements of the three packs (b_i, for i = 1 to 3) are shown below.

	Packs		
	1	2	3
Pack 1	-	4	4

$b_i = 10$; $b_2 = 10$, $b_3 = 9$

The algorithm searches the pack 1 row to find the smallest entry which corresponds to the most similar, unassigned pack type. Obviously, a tie exists (or does it?).

Pack 2 requires ten different component types. From the definition of the term similar, we know that six of those different component types are also required by pack 1, since 4 is the entry under pack 2, indicating that four additional component types must be mounted to produce packs 1 and 2 together. Pack 3 requires nine different component types. Five of those are also required by pack 1, for the same reasons presented with pack 2. These operations demonstrate that six of ten, or 60% of the different component types required by pack 2 are also required by pack 1. Only five of nine, or 55.6 of the different component types required by pack 3 are also required by pack 1. The higher percentage indicates a higher degree of similarity.

Therefore, although Array DIFFER reflects the same degree of similarity between pack 1 and packs 2 and 3 in Example 4.2, these additional computations actually reveal that pack 2 is, in fact, a little more similar to pack 1 than pack 3. This tie-breaking procedure, termed the Highest Degree of Similarity procedure, or HDOS, is used every time the algorithm attempts to select the most <u>similar</u>, unassigned pack type for a particular production run, and Array DIFFER indicates that a tie exists. It is possible that even after these computations are performed, a bonafide tie still exists. In this case, the tie is broken arbitrarily, since the degree of similarity is actually identical.

Before the al orithm assigns this second pack type to the k^{th} sequencer, it verifies that the assignment of this pack type will not violate the dispensing head limit (C_k), nor volume capacity limit (S_k), of the k^{th} sequencer. The algorithm adds the current number of allocated heads (H_k) to the number of additional component tapes required to be mounted to produce the selected pack type (b_{iq}). If

$$H_k + b_{iq} \leq C_k, \qquad (4)$$

then sufficient, unallocated heads are available to produce this second pack type on the same production run.

The algorithm adds the current production value requirement (PVR_k) to the production value requirement of the selected pack type (PVR_i). If

$$PVR_k + PVR_i \leq S_k, \qquad (5)$$

then the volume capacity for the k^{th} sequencer will not be exceeded if this second pack type is assigned to that sequencer on the same production run. If the assignment of this second pack type to the k^{th} sequencer will, in fact, violate either the dispensing head limit (C_k), or the volume capacity limit (S_k), the algorithm does not assign a second pack type to sequencer k, and begins scheduling sequencer k + 1 (for k = 1 to L - 1). This particular event is exceptionally remote, especially in light of the assumptions considered during the development of this heuristic approach. However, the method by which the algorithm treats this peculiar event is addressed in Section 4.3.4.

If neither the dispensing head limit (C_k), nor the volume capacity limit (S_k), is violated, the algorithm assigns this second pack type for production to the k^{th} sequencer. Immediately following this assignment, the algorithm performs its two customary bookkeeping operations (Section 4.3.2). The procedure used to increment the production volume requirement (PVR_k) of the k^{th} sequencer is identical to that described in Section 4.3.2. The production volume requirement of the second assigned pack type (PVR_i) is added to the previous production volume requirement of the k^{th} sequencer (PVR_k), to yield a current value of PVR_k, which reflects the assignment of the second pack type.

$$PVR_{k,current} = PVR_{k,previous} + PVR_i \qquad (6)$$

This incremental procedure, which updates the current production volume requirement (PVR_k) of the k^{th} sequencer, is utilized identically throughout the entire algorithm, regardless of whether one, or two, or more than two pack types are assigned to the k^{th} sequencer.

The deviation in the bookkeeping operation which computes the current number of allocated dispensing heads (H_k) for the k^{th} sequencer occurs when the second, and multiple pack types are assigned. When the first pack type is assigned to the kth sequencer, the current number of allocated dispensing heads (H_k) is exactly equal to the total number of different component types required to produce that first assigned pack type (refer to equation (2)). Whenever two, or more pack types are assigned to the kth sequencer on a particular production run, the current number of allocated dispensing heads (H_k) equals the sum of the previous value of H_k, and the number of additional component tapes required to be mounted (b_{iq}) to produce the most recently assigned pack type i.

$$H_{k,\text{current}} = H_{k,\text{previous}} + b_{iq} \qquad (7)$$

This incremental procedure, which updates the current number of allocated dispensing heads (H_k) on the kth sequencer when two, or more pack types are assigned, is utilized identically throughout the entire algorithm. The deviation in this bookkeeping operation surfaces only when the first pack type is assigned to the kth sequencer, in which case H_k is calculated by equation (2).

4.3.4 Comparison Operation Following Assignment of Second Pack Type

Once the first and second pack types are assigned for production to any of the L sequencers, the algorithm again searches for a most similar, unassigned pack type. The first M rows of Array DIFFER are of no use in this situation. The M*M portion of DIFFER contains numerical entries indicating the degree of similarity between any two pack types, of which only one is assigned. Obviously, when two, or more pack types are previously assigned to the kth sequencer, those entries are meaningless.

However, in keeping with the strategy of this heuristic approach, the algorithm must be capable of locating the unassigned pack type that is most similar to the aggregation of pack types previously assigned to the kth sequencer, for a particular production run. The last L rows of the total M+L rows in Array DIFFER provide the algorithm with a tool to accomplish this task. The entries in the last L rows of DIFFER are computed only after the algorithm assigns two, or more, pack types to the kth sequencer for a particular production run.

The numerical entries in the M+kth row (for k = 1 to L) of Array DIFFER indicate the degree of similarity between each, unassigned pack type, and the aggregation of pack types previously assigned to the kth sequencer, for a particular production run. In other words, the numerical entries in the M+kth row represent the comparison of the different component types required by each, unassigned pack type against the union of different component types required to be mounted on the kth sequencer in order to produce all of the pack types previously assigned to that sequencer, for a particular production run.

The method by which the numerical entries in the last L rows of Array DIFFER are generated is through a procedure involving a specific comparison operation performed by the algorithm. The first step in this procedure is to represent the union of different component types required to produce all previously assigned pack types on the kth sequencer, for a particular production run, in vector form. The current number of allocated dispensing heads (H_k) indicates the total number of different component types required to produce all of the assigned pack types together. Recall that the input parameter N is the total number of different component types. The algorithm establishes a one-dimensional vector, which is N elements long. This vector, called the U vector, is initialized with zeroes.

The algorithm scans the different component types required by all pack types previously assigned to the kth sequencer for a particular production run. Let Q represent the set of pack types which are already assigned to sequencer k. If $a_{ij} = 1$ (for i in set Q; j = 1 to N), the algorithm places a value of 1 in the jth element of the U vector. By this method, the H_k different component types required to produce all previously assigned pack types are represented by the U vector. The only elements in the U vector with a value of 1 are those elements corresponding to the different component types required by at least one of the pack types assigned to sequencer k.

Once the U vector is established, corresponding specifically to sequencer k for a specific production run, all remaining, unassigned pack types are compared against it. In other words, the different component types required to produce each, unassigned pack type are compared against the different component types represented in the U vector. This comparison operation is performed in a manner similar to the comparison operation which generates the numerical entries in the first M rows of Array DIFFER. Refer to Section 4.2.2.

The results of this comparison operation yield the numerical entries in the M+kth row of Array DIFFER (corresponding to the kth sequencer for which the comparison was conducted). This comparison operation is conducted every time another pack type is assigned to the kth sequencer, starting with the second pack type. Therefore, the numerical entries in the M+k^{th} row (for k = 1 to L) of DIFFER indicate the degree of similarity between each, unassigned pack type, and the current aggregation of pack types previously assigned to the k^{th} sequencer for a particular production run. In this manner, the algorithm is always capable of selecting the most similar, unassigned pack type as it schedules the L available sequencers.

The only case in which the algorithm does not perform this comparison operation as it schedules the k^{th} sequencer is in the remote event that the first assigned pack type is the only pack type that can be assigned to the k^{th} sequencer for a particular production run. In this event, since a second pack type is not assigned to the k^{th} sequencer, the numerical entries in the M + k^{th} row of DIFFER would not be needed for this first part of the algorithm; hence are not generated by the comparison operation. However, since the second part of the algorithm also utilizes the last L rows of DIFFER, we must calculate these entries any way.

Recall that the numerical entries in the first M rows of DIFFER indicate the degree of similarity between any two pack types. Therefore, to generate numerical entries for the M + k^{th} row of DIFFER, the numerical entries for the M+k^{th} row of DIFFER, given that this remote event occurs, the numerical entries in the i^{th} row of the M*M portion of DIFFER (corresponding to the single, first assigned pack type i) are copied directly into the M+k^{th} row of DIFFER (corresponding to the k^{th} sequencer to which the single pack type i is assigned).

4.3.5 Assignment of Remaining Pack Types to Sequencers

The M + k th row of Array DIFFER enables the algorithm to repeatedly select the most similar, unassigned pack type as it schedules the k th sequencer for a particular production run. The algorithm locates the most similar, unassigned pack type by specifically searching the M + k th row of DIFFER, for k = 1 to L (corresponding to the k th sequencer currently being scheduled). The algorithm selects the smallest numerical entry corresponding to an unassigned pack type (0 is a possible entry). This smallest numerical entry represents the unassigned pack type that is most similar to the aggregation of all pack types previously assigned to the k th sequencer for that particular production run.

The term for the individual numerical entries in the M + k th row is b_{kq} (for k = 1 to L; q = 1 to M). The definition of b_{kq} is identical to the definition of b_{iq} presented in Section 4.3.3, with one exception. This exception is that b_{kq} indicates the degree of similarity between the aggregation of all pack types previously assigned to the k th sequencer for a particular production run, and pack type q, which is unassigned. Recall that b_{iq} indicates the degree of similarity only between pack type i and pack type q.

Two items specific to the last L rows, or L*M portion, of Array DIFFER, are briefly addressed. First, unlike the i th row (for i = 1 to M), the M + k th row (for k = 1 to L) of DIFFER does not contain M-1 numerical entries. In fact, the total number of entries in any given M + k th row generated by the comparison operation for that corresponding k th sequencer is inversely proportional [illegible] number of pack types assigned to <u>all</u> <u>sequencers</u> over <u>all</u> <u>production runs</u>. The reason for this rather unremarkable occurrence is simple. A numerical entry in the q^{th} column (for q = 1 to M) of the $M + k^{th}$ row (for k = 1 to L) corresponds to an unassigned pack type q. As the total number of assigned pack types increases, the total number of unassigned pack types decreases, thereby decreasing the total number of elements containing numerical entries.

The second item addressed is the method by which the algorithm breaks a tie when multiple unassigned pack types appear to be <u>most similar</u> to the current aggregation of previously assigned pack types (multiple b_{kq}'s in the $M + k^{th}$ row). The algorithm implements the Highest Degree of Similarity (HDOS) procedure by performing the previously described computations (Example 4.2) on the unassigned pack types with a value of b_{kq} in that particular $M + k^{th}$ row of [illegible]. Again, if a tie still exists after the HDOS procedure is performed, the tie is broken arbitrarily.

Before the algorithm assigns an additional pack type to the k^{th} sequencer, it verifies that the assignment of the pack type will not violate the dispensing head [illegible] [illegible] It pre-computes the two [illegible] allocated dispensing heads [illegible] the k^{th} sequencer.

[illegible]

[illegible]

If neither of these two equations is violated, the algorithm assigns the pack type for production to the kth sequencer. Immediately following this assignment, the algorithm performs its two customary bookkeeping operations that we presented through equations (6) and (7) (Section 4.3.2). We repeat these equations here for ease of reference:

$$PVR_{k,current} = PVR_{k,previous} + PVR_q \tag{6}$$

$$H_{k,current} = H_{k,previous} + b_{kq} \tag{7}$$

The algorithm then performs the comparison operation to update the numerical entries in the appropriate M + kth row of DIFFER.

The procedures described in this section up to this point are performed repeatedly, for k = 1 to L, until the algorithm may no longer assign additional pack types to the Lth sequencer. As mentioned earlier, the nature of the component type requirements of the pack types to be produced might be such that all pack types are scheduled for production in the first part of the algorithm. In this case, the sequenced tape production schedule is completed, and the second part of the heuristic is never invoked. On the other hand, if at this time there still are some pack types which are not yet assigned to the sequencers for production, the algorithm proceeds with its second part.

When the first part is terminated, the scheduling process is in the following state:

1. [illegible] sequencers are scheduled for one production run only, and are assigned to produce at least one sequenced tape. The M + kth row of array DIFFER, for k = 1 to L, indicates the degree of similarity

between each, unassigned pack type, and the current aggregation of all pack types assigned to the k th sequencer for the first production run. PVR_k (for $k = 1$ to L) indicates the total number of component insertions that the k th sequencer must make in order to produce all of its assigned corresponding sequenced tapes for the first production run. H_k (for $k = 1$ to L) is reset to 0. The remaining sections of this chapter pertain to the second part of the heuristic algorithm.

4.4 Heuristic Algorithm - Part Two

The sequenced tape assignment process in this second part resembles much of what is previously outlined in the first part. The purpose of this second part is to schedule all unassigned pack types for production. The detailed description of this second part is decomposed into five distinct segments. The first segment describes the method by which the available sequencers are selected to make multiple production runs. The second part describes the method by which the first pack type is assigned to the selected sequencer. The third segment describes two bookkeeping operations performed by the algorithm after the first pack type is assigned. The fourth segment describes the method by which the second pack type is assigned for production and describes the comparison operation performed by the algorithm after the second pack type is assigned. The fifth segment describes the method by which remaining, unassigned pack types are repeatedly assigned to the sequencers until all pack types are scheduled, in which case the sequenced tape production schedule is completed, and the algorithm terminates.

There is one unique aspect to this second part. Every time a pack type is assigned for production, the algorithm performs the customary bookkeeping operations and immediately determines if any unassigned pack types remain. In this manner, the algorithm may terminate as soon as possible.

4.4.1 Selection of Sequencers for Multiple Production Runs

In this second part, the algorithm selects a sequencer to undergo scheduling based on its current production volume requirement ($PVR_{k,current}$). The algorithm always selects that sequencer k (for k = 1 to L) that is required to make the fewest total number of component insertions up to that stage in the planning horizon in order to produce all of its previously assigned pack types over all production runs.

The logic behind this selection rule stems from the second of management's two goals--maintain a relatively balanced workload distribution. By selecting the sequencer with the smallest incurred workload to make the next production run, the algorithm does not permit any sequencer to stray too far from the other available sequencers in terms of the total number of component insertions.

4.4.2 Assignment of the First Pack Type to Sequencers

There exists only one rule governing the selection of the first pack type assigned to the sequencer with the smallest production volume requirement (PRV_k). This rule, the 'most common' rule, is in keeping with the strategy of the heuristic approach, and is discussed in Chapter 5. Pack types have already been scheduled on each of the L available sequencers in the first part. The algorithm attempts to assign pack types together on the same sequencer based on their high degree of similarity.

Therefore, the algorithm locates the <u>most</u> <u>common</u>, unassigned pack type by specifically searching the M + k^{th} row of DIFFER, for k = 1 to L (corresponding to the k^{th} sequencer currently being scheduled). Obviously, one of the definitions of <u>common</u> corresponding to one of the three different heuristic procedures affects the method by which the algorithm selects the unassigned first pack type. By searching the M + k^{th} row of DIFFER, the algorithm locates the unassigned pack type that is most <u>common</u> to the current aggregation of all pack types on the k^{th} sequencer assigned for the production run most recently scheduled. If a tie exists, the HDOS procedure is implemented as described in Section 4.3.5. If a tie still exists after that, it is broken arbitrarily.

4.4.3 <u>Bookkeeping Operations After Assignment of Pack Types</u>

The two bookkeeping operations performed after the first pack type is assigned for production are also performed in the first part. The production volume requirement (PVR_k) of the k^{th} sequencer being scheduled is a cumulative total over all production runs, and is incremented by the production volume requirement of the first assigned pack type i. Equation (6), Section 4.3.3, is utilized to update the current total workload:

$$PVR_{k,current} = PVR_{k,previous} + PVR_i \qquad (6)$$

The number of allocated dispensing heads (H_k) is incremented from 0 to the total number of different component types (b_i) required to produce that first assigned pack type i. Equation (2), Section 4.3.2, is utilized to update H_k:

$$H_{k,\text{current}} = b_i \tag{2}$$

Immediately after computing $PVR_{k,\text{current}}$ and $H_{k,\text{current}}$, the algorithm determines if any pack types are still unassigned. If this is the case, the algorithm continues. If not, the algorithm terminates.

4.4.4 Assignment of the Second Pack Type to Sequencers and the Following Comparison Operation

The operations performed to select the second pack type assigned to the k^{th} sequencer being scheduled are identical to the operations described in Section 4.3.3, with one exception. In this second part, only equation (4) is pre-computed to determine if a second pack type may be assigned. The reason why equation (5) is not pre-computed is because a sequencer volume capacity limit (S_k) is not imposed on the sequencers in this second part. The sequencer being scheduled is selected based on its smallest total production volume requirement (PVR_k) in comparison to the other sequencer production volume requirements.

This second part does not guarantee that every sequencer will make the same number of production runs, as in the first part. At some point in time, all of the pack types will eventually be scheduled for production. Because of this uncertainty, the sequencers are permitted to produce as many sequenced tapes as possible until an insufficient number of unallocated heads exists, at which time the algorithm selects the next sequencer for production, if required.

As usual in this second part, the algorithm updates the number of allocated dispensing heads (H_k), and the production volume requirement (PVR_k). The algorithm then determines if any unassigned pack types still remain. If unassigned pack types still remain, the algorithm performs the comparison operation as described in Section 4.3.4.

4.4.5 Assignment of Remaining Pack Types to Sequencers

The M + kth row of Array DIFFER enables the algorithm to repeatedly select the most similar, unassigned pack type as it schedules the kth sequencer for a multiple production run. The process by which the algorithm selects the remaining, unassigned pack types, and selects the sequencer to be scheduled for multiple production runs in this second part, is as described in Sections 4.4.1 through 4.4.4.

The algorithm repeatedly performs these operations on the kth sequencer (for k = 1 to L) until either all pack types are assigned, or until the dispensing head limit (C_k) is approached, but not violated. In the case where the dispensing head limit (C_k) prevents additional pack types from being assigned to the kth sequencer, and not all pack types have been assigned, the algorithm selects some sequencer k for another production run based on its smallest incurred production volume requirement (PVR_k). In the case where all pack types are assigned, the production schedule is completed and the algorithm terminates.

CHAPTER 5

THREE HEURISTIC PROCEDURES

5.1 COMMON Pack Types

The three heuristic procedures developed to provide a solution to the sequencer scheduling/assignment problem differ in only one respect. Each heuristic procedure enlists a distinct definition of the term common. Otherwise, the specific algorithm followed by the three heuristic procedures is identical to that described in Chapter 4. The purpose of manifold definitions of the term common serves to establish different relationships between the component type requirements of the individual pack types.

Recall that the algorithm adheres to the strategy of assigning similar pack types to the same sequencer. It separates pack types with little in common, in terms of component type requirements, by assigning them to different sequencers. It is previously explained in Section 4.2.1 that, with the exception of the very first pack type selected by the starting rule, the first pack type assigned to the k^{th} sequencer (for k = 1 to L), for all production runs, is selected by incorporating one of three definitions of common. The method by which the first assigned pack type is selected in the first part of the algorithm is via the 'least common' rule. The logic supporting this rule is discussed in Section 4.3.1. The method by which the first assigned pack type is selected in the second part of the algorithm is via the 'most common' rule. The logic supporting this rule is discussed in Section 4.4.2. To emphasize, the only situation where the term common is incorporated is when the algorithm is selecting the first pack type assigned to sequencers in both parts.

All three definitions of the term common are related to the definition of the term similar. The operations conducted to determine the three common relationships between pack types are performed simply by manipulating the numerical entries indicating the degree of similarity between pack types in Array DIFFER. The three definitions of common are explained in the following sections and the operations performed to determine the common relationship between pack types is described. The final section of this chapter demonstrates the methods by which the 'least common' rule is implemented.

5.2 Definitions of COMMON

As mentioned earlier, the three definitions of common are related to the definition of similar. The precise definition of the term similar is presented in Section 4.2.1. In short, the degree of similarity between an unassigned pack type and the assigned pack type(s) is determined by the number of additional, different component tapes that must be mounted to produce all of the pack types together. An unassigned pack type is most similar to the aggregation of assigned pack types on a particular sequencer if the number of additional, different component tapes that must be mounted to produce them together is the fewest number possible.

5.2.1 Most SIMILAR, Most COMMON

The first definition of common is the identical definition of similar. This definition of common is related to the number of additional, different component tapes that must be mounted in order to produce each, unassigned pack type together with previously assigned pack types.

When the algorithm searches for the least common pack type in the first part, it searches over all previously scheduled sequencers to locate the least common, unassigned pack type with respect to the component type requirements of all previously assigned pack types. The method by which the algorithm is capable of doing this is described later in Section 5.3.

When the algorithm searches for the most common pack type in the second part, it simply searches the appropriate $M + k^{th}$ row (corresponding to the k^{th} sequencer selected to be scheduled) which reflects the current aggregation of all pack types previously assigned for the most recently scheduled production run.

With this definition of common, the smallest numerical entry in the $M + k^{th}$ row represents the most common, unassigned pack type, while the largest numerical entry represents the least common, unassigned pack type. If the algorithm locates multiple numerical entries that qualify as least common in the first part, or most common in the second part, the HDOS procedure is implemented. If a tie still exists, it is broken arbitrarily. This definition of common is incorporated in the heuristic procedure named YFIX.

5.2.2 Least SIMILAR, Most COMMON

The second definition of common is the exact inverse of the definition of similar. This definition of common is related to the number of different component types that are required by each, unassigned pack type and by the aggregation of previously assigned pack types.

When the algorithm searches for the least common pack type in the first part, it searches over all previously scheduled sequencers with respect to the component type requirements of all previously assigned pack types. When the algorithm searches for the least common pack type in the second part, it simply searches the M + k th row (corresponding to the k th sequencer selected to be scheduled).

With this definition of common, the numerical entries generated by the comparison operation must be manipulated to establish the correct common relationship. The algorithm subtracts the entries in the elements of the appropriate M + k th row of DIFF- corresponding to unassigned pack types from the total number of different component types (b_i) required by each, unassigned pack type. The resulting values indicate the number of different component types required by each, unassigned pack type which are also required by the aggregation of pack types previously assigned. These resulting values are inserted back into the elements of the M + k th row.

With this definition of common, the smallest numerical entry in the M + k th row represents the least common, unassigned pack type, while the largest numerical entry represents the most common, unassigned pack type. If the algorithm locates multiple numerical entries that qualify as least common in the first part, or most common in the second part, the HDOS procedure is implemented. If a tie still exists, it is broken arbitrarily. This definition of common is incorporated in the heuristic procedure named V2YFIX.

5.2.3 COMMON Pack Types Determined as Proportions

This final definition of common seeks to eliminate an inherent deficiency in the first two definitions of common. The deficiency in those two definitions is that the algorithm searches specifically for either the largest or smallest numerical entry in the M + k^{th} row to locate either the most or least common unassigned pack type, depending on whether the algorithm is in its first or second part.

These two definitions do not take into account the total number of different component types required by each unassigned pack type (b_i). It is this precise deficiency that requires the existence of a tie-breaking procedure such as the HDOS procedure. This deficiency is demonstrated by an example.

EXAMPLE 5.1

Suppose that the scheduling process is in the following state:

L = 1 available sequencer.

M = 4 pack types to be produced.

N = 12 total different component types.

C_k = 10 dispensing heads.

Assume that the YFIX heuristic procedure is being used, and the algorithm just terminated the first part. In that first part, it assigned two pack types (packs 1 and 3) for production. The total number of different component types (b_i) required by the four pack types are 7, 5, 6, and 9, respectively. The numerical entries in the M + k^{th} row generated by comparison operation are as follows:

	Packs			
	1	2	3	5
M + kth Row	-	2	-	3

The algorithm initiates the second part by scheduling the one available sequencer for a second production run. The algorithm searches the M + kth row to locate the most common, unassigned pack type in order to assign that pack type first for the second production run. Hash marks are in the row elements of pack types 1 and 3 because they were assigned for production in the first part. Since the YFIX heuristic procedure is being used, the most common, unassigned pack type is that pack type with the smallest numerical entry in the M + kth row. Manipulation of the entries in the M + kth row are not required because the YFIX definition of common is identical to the definition of similar.

Therefore, the algorithm selects pack type 2 as most common, since it is the unassigned pack type with the smallest numerical entry (2). Some computations reveal, however, that pack type 4 is really more 'common' to the aggregation of pack types 1 and 3.

Since 2 is the entry corresponding to pack type 2, this indicates that three component types required by pack type 2 are also required by the aggregation of pack types 1 and 3 together. Therefore, three of five, or 60% of the component types required by pack type 2 are also required by pack types 1 and 3 together.

Since 3 is the entry corresponding to pack type 4, this indicates that six component types required by pack 4 are also required by the aggregation of pack types 1 and 3 together. Therefore, six of nine, or 67% of the component types required by pack type 4 are also required by pack types 1 and 3 together.

It is evident that pack type 4 has a higher proportion of different component types in 'common' with pack types 1 and 3 together. However, the algorithm does not consider this situation with either the YFIX or V2YFIX definitions of common.

Therefore, the final definition of common determines the proportion of component types required by each, unassigned pack type with respect to its total number of different component types (b_i).

The algorithm performs this manipulation simply by dividing the i^{th} entry of the $M + k^{th}$ row by the total number of different component types required by each pack type i, b_i, for each unassigned pack type, i = 1 to M. The algorithm inserts these proportion results back into the $M + k^{th}$ row. With this definition of common, the smallest proportion in the $M + k^{th}$ row represents the most common, unassigned pack type, while the largest proportion in the $M + k^{th}$ row represents the least common, unassigned pack type.

The reason why the smallest proportion represents the most common, unassigned pack type is because the numerical entries in the $M + k^{th}$ row generated by the comparison operation indicate the number of different component tapes that must be additionally mounted to produce each, unassigned pack type, and the aggregation of previously assigned pack types together. This definition of common is incorporated in the heuristic procedure named PROYFIX. In the PROYFIX heuristic procedure, the HDOS procedure is not necessary. If the algorithm locates multiple proportions that qualify as least common in the first part, or most common in the second part, the tie is broken arbitrarily.

5.3 Implementation of the Least Common Rule

The least common rule permits the algorithm to select the first pack type assigned to the k^{th} sequencer for k = 2 to [illegible], in the first part, in full accordance with the strategy of the heuristic approach. The least common pack type is selected with respect to all previously assigned pack types on all previously assigned sequencers. This procedure is accomplished through the use of a one-dimensional array, named Array HOLD. The operations performed to utilize Array HOLD, which is M elements long, are identical regardless of which definition of common is incorporated.

After the algorithm completely schedules the first sequencer, the numerical entries in the M + 1st row, generated by the comparison operation, indicate the degree of similarity between each, unassigned pack type and the final aggregation of all pack types assigned to that first sequencer. These numerical entries are manipulated accordingly, based on the heuristic procedure being utilized, and inserted into the corresponding elements of Array HOLD.

To locate the least common, unassigned pack type with respect to the final aggregation of pack types assigned to the first sequencer, the algorithm specifically searches Array HOLD. Once it locates the least common, unassigned pack type, the algorithm assigns it to the second sequencer, and performs the operations described in Sections 4.3.1 through 4.3.5 to completely schedule the second sequencer. The comparison operation generates numerical entries in the M + 2nd row indicating the degree of similarity between each, unassigned pack type, and the final aggregation of all packs assigned to the second sequencer.

To locate the least common, unassigned pack type, for assignment to the third sequencer, with respect to the final aggregation of pack types previously assigned to the first and second sequencers, the algorithm would have to search Array HOLD and the M + 2nd row of DIFFER, whose numerical entries are appropriately manipulated based on a definition of common.

The algorithm avoids this situation, however, by comparing the corresponding numerical entries in Array HOLD and the M + 2nd row of DIFFER. The algorithm compares the numerical entries in the elements corresponding to the unassigned pack types, and locates that numerical entry that is indicated to be the most common of the two. The algorithm inserts the most common numerical entry in the corresponding element of Array HOLD. Thus, Array HOLD now contains the numerical entries indicating the most common relationship between each, unassigned pack type, and the final aggregation of previously assigned pack types over the two previously scheduled sequencers.

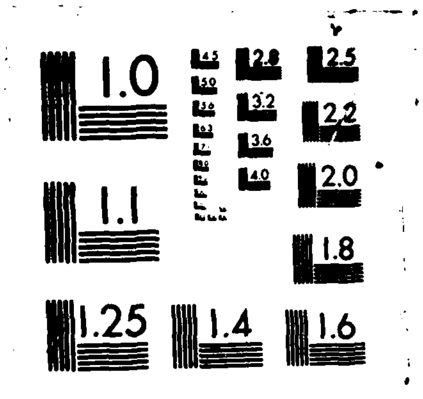
The algorithm then searches Array HOLD and locates the least common, unassigned pack type indicated by the individual numerical entries. In this manner, the algorithm, in fact, selects the least common, unassigned pack type to be scheduled first on [illegible] sequencer.

This procedure is performed every time the algorithm [illegible] schedules a sequencer, until all [illegible] available [illegible]

AD-A183 935 PRODUCTION SCHEDULING OF SEQUENCED TAPES FOR PRINTED CIRCUIT PACK ASSEMBLY(U) ARMY MILITARY PERSONNEL CENTER ALEXANDRIA VA PROGRAM ANALYSIS OFFICE S J RAWLICK 2/2

UNCLASSIFIED 09 JUL 87 F/G 9/1 NL

END
9-87
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

Array HOLD always contains the most common numerical entries over all previously assigned pack types over all previously scheduled sequencers. By locating the least common, unassigned pack type in Array HOLD, the algorithm is, in fact, locating the least common pack type with respect to all previously assigned pack types, over all previously scheduled sequencers.

CHAPTER 6

NUMERICAL RESULTS AND CONCLUSIONS

Chapters 3 through 5 provide a detailed description of the sequencer scheduling assignment problem, and our proposed heuristic procedure for solving it. In this chapter we present the results of some numerical experiments with these procedures.

This specific variation of the sequenced tape production scheduling problem is very complex in nature, and as such, is not even particularly well defined. We stated earlier that an 'optimal' solution to this problem would be difficult to obtain, even for the relatively simple case where only one sequencer is involved. The case with more than one sequencer is increasingly more difficult.

Ideally, we would like to compare the schedules obtained using this heuristic approach on a particular set of problems with their respective 'optimal' schedules. In the absence of a methodology to obtain, or even properly define, such an optimal schedule, however, it would not be possible for us to do so. Hence, in this study, we limit our discussion to a subjective analysis of the performance of this approach.

The measures of goodness described in Chapter 3 provide some reference by which different heuristic procedures may be compared with each other. These measures of goodness serve to gauge the relative merits of the three heuristic procedures, but they are subject to interpretation. It cannot be accurately stated, for instance, that a production schedule requiring each of three sequencers to make two change-overs apiece is categorically better, or worse, than another production schedule, devised for the same problem, which requires that the same three sequencers each make one, two, and three change-overs, respectively.

The nature of the sequencer scheduling/assignment problem is such that it does not have a readily apparent structure. It is for this precise reason that the development of an approach which provides a satisfactory solution is so frustrating. The purpose of this heuristic approach is to create an adequate form of structure by which a sensible solution may be obtained.

The framework of the experiment phase incorporating the three heuristic procedures reflects this lack of analytical certainty. The method by which the heuristic procedures are validated serves to demonstrate that the algorithm described in Chapter 4, in fact, functions very nicely, and that the resulting production schedules devised for numerous problems are reasonable in light of management's two production goals. Some trends are observed as various sets of input parameters are introduced.

The computer programs for the three heuristic procedures are written in FORTRAN, and run on a VAX 11/750 computer. A listing of the YFIX heuristic procedure is included in Appendix 8.2. The simulated data set pertaining to the pack type component requirements closely resembles actual data for a production environment involving two sequencers.

The next two sections contain discussion related to the experiment phase of the three different heuristic procedures, and present summary computational results.

6.1 Typical Problems in a Sequencing Environment

The set of simulated data used in conjunction with the numerical experiments is typical of a sequencing environment in which the total number of different component types required by each pack type, in most cases, does not exceed 60 (b_i, for i = 1 to M). In fact, a good number of pack types in this data set require a relatively low number of different component types (less than 20). For this reason, no more than two sequencers are generally required for the production of their corresponding sequenced tapes.

Six typical problems are solved, with the number of pack types to be produced ranging in number from 20 to 40. These six problems are solved by all three heuristic procedures. The number of dispensing heads on the two available sequencers (C_k) is originally fixed at 60. When a resulting production schedule requires a change-over at this figure, C_k is increased to 100 (a more typical limit). This type of reaction to imminent sequencer change-overs realistically reflects the position of management in their quest for efficient production operations.

Additionally, the sequencer volume capacity (S_k) of both sequencers is calculated according to equation (1) in Section 2.3. Throughout these numerical experiments, we calculate the 'constant' term in that equation as a percentage of the total production volume. We let this percentage vary from 5% to 25% in solving the six problems. This parameter is altered to empirically observe the degree of sensitivity of the heuristic procedures to these volume constraints.

The numerical results are presented in Tables 6.1 through 6.6--one table for each problem. The key to deciphering the tables is described as follows.

The upper left hand corner indicates the number of available sequencers (L); and the total number of pack types scheduled for production (M). The number of dispensing heads fixed for each particular problem is listed in the C_k column alongside the name of the heuristic procedure used to solve that problem. Three different percentages used in the calculation of S_k are presented in the column headed 'S_k.' A value of 5% restricts the problem, while a value of 25% relaxes the problem. The five measures of goodness described in Chapter 3 are presented in columns 1 through 5 under 'Measures of Goodness.'

The total number of sequencer change-overs for a given problem is represented in column 1. The total number of component tape changes required by all sequencers in a given problem is given in column 2. The total sequencer production volume requirements are in column 3, with each entry corresponding to one sequencer. The multi-tiered entries in column 4 represent the total number of different component types required by each sequencer to produce all of their assigned sequenced tapes. Each entry corresponds to one sequencer. The multi-tiered entries in column 5 represent the total number of 'dedicated' heads on each sequencer. A block marked by a single hash mark indicates that all available sequencers required only one production run to produce all of their assigned sequenced tapes. Therefore, all of the allocated heads could be considered as

Table 6.1 Problem 1

L=2	M = 20		Measures of Goodness						CPU
	C_k	S_k	1	2	3		4	5	(sec)
		5%	0	0	158	180	13 35	-	2.81
YFIX	60	10%	0	0	202	136	19 28	-	2.52
		25%	0	0	248	90	29 23	-	2.54
		5%	0	0	158	180	13 35	-	2.51
V2YFIX	60	10%	0	0	202	136	19 28	-	2.45
		25%	0	0	248	90	29 23	-	2.59
		5%	0	0	158	180	13 35	-	2.48
PROYFIX	60	10%	0	0	202	136	19 28	-	2.54
		25%	0	0	248	90	29 33	-	2.58

Table 6.2 Problem 2

L=2	M = 20		Measures of Goodness					CPU
	C_k	S_k	1	2	3	4	5	(sec)
YFIX	60	5%	1	17	468 280	53 / 37	10 / -	2.59
		10%	0	0	412 336	46 / 50	-	2.61
		25%	0	0	412 336	46 / 50	-	2.65
V2YFIX	60	5%	1	22	416 332	58 / 35	5 / -	2.57
		10%	0	0	412 336	46 / 50	-	2.67
		25%	0	0	412 336	46 / 50	-	2.60
PROYFIX	60	5%	1	17	468 280	53 / 37	10 / -	2.65
		10%	0	0	412 336	46 / 50	-	2.58
		25%	0	0	412 336	46 / 50	-	2.63
YFIX	100	5%	1	17	468 280	53 / 37	10 / -	2.60
V2YFIX	100	5%	1	22	416 332	58 / 35	5 / -	2.61
PROYFIX	100	5%	1	17	468 280	53 / 37	10	2.59

Table 6.3 Problem 3

L=2	M = 30		Measures of Goodness					CPU
	C_k	S_k	1	2	3	4	5	(sec)
		5%	0	0	538 472	50 45	-	5.96
YFIX	60	10%	0	0	538 472	50 45	-	5.86
		25%	0	0	538 472	50 45	-	5.89
		5%	0	0	538 472	50 45	-	5.94
V2YFIX	60	10%	0	0	538 472	50 45	-	5.90
		25%	0	0	538 472	50 45	-	5.93
		5%	0	0	538 472	50 45	-	5.89
PROYFIX	60	10%	0	0	538 472	50 45	-	5.93
		25%	0	0	538 472	50 45	-	6.00

Table 6.4 Problem 4

L=2	M = 30		Measures of Goodness					CPU
	C_k	$\cdot$ S_k	1	2	3	4	5	(sec)
		5%	1	31	498 892	58 89	- 16	5.75
YFIX	60	10%	1	31	498 892	58 89	- 16	5.77
		25%	1	31	498 892	58 89	- 16	5.74
		5%	2	30	640 750	70 74	15 9	5.77
V2YFIX	60	10%	2	30	640 750	70 74	15 9	5.77
		25%	1	31	498 892	58 89	- 16	5.75
		5%	1	31	498 892	58 89	- 16	5.75
PROYFIX	60	10%	1	31	498 892	58 89	- 16	5.76
		25%	1	31	498 892	58 89	- 16	5.71

continued

Table 6.4 (continued)

L=2	M = 30		Measures of Goodness						CPU
	C_k	S_k	1	2	3		4	5	(sec)
		5%	0	0	722	668	77 76	-	6.08
YFIX	100	10%	0	0	722	668	77 76	-	5.83
		25%	0	0	1024	366	94 25	-	5.97
		5%	0	0	722	668	77 76	-	6.00
V2YFIX	100	10%	0	0	722	668	77 76	-	6.03
		25%	0	0	1024	366	94 25	-	6.13
		5%	0	0	722	668	77 76	-	6.15
PROYFIX	100	10%	0	0	722	668	77 76	-	5.85
		25%	0	0	1024	366	94 25	-	5.95

Table 6.5 Problem 5

L=2	M = 40		Measures of Goodness					CPU
	C_k	S_k	1	2	3	4	5	(sec)
		5%	1	14	1010 1106	59 74	- 29	10.45
YFIX	60	10%	1	14	1010 1106	59 74	- 29	10.39
		25%	1	14	1010 1106	59 74	- 29	10.37
		5%	1	19	1010 1106	59 74	- 35	10.52
V2YFIX	60	10%	1	19	1010 1106	59 74	- 35	10.45
		25%	1	19	1010 1106	59 74	- 35	10.40
		5%	1	19	1010 1106	59 74	- 35	10.38
PROYFIX	60	10%	1	19	1010 1106	59 74	- 35	10.30
		25%	1	19	1010 1106	59 74	- 35	10.32

continued

Table 6.5 (continued)

L=2	M = 40		Measures of Goodness						CPU
	C_k	S_k	1	2	3		4	5	(sec)
		5%	0	0	1076	1040	72 69	-	10.65
YFIX	100	10%	0	0	1076	1040	72 69	-	10.71
		25%	0	0	1572	544	88 55	-	10.78
		5%	0	0	1076	1040	72 69	-	10.43
V2YFIX	100	10%	0	0	1076	1040	72 69	-	10.59
		25%	0	0	1572	544	88 55	-	10.78
		5%	0	0	1076	1040	72 69	-	10.57
PROYFIX	100	10%	0	0	1076	1040	72 69	-	10.46
		25%	0	0	1572	544	88 55	-	10.84

Table 6.6 Problem 6

L=2	M = 40		Measures of Goodness						CPU
	C_k	S_k	1	2	3		4	5	(sec)
		5%	2	19	1008	1020	66 67	31 24	10.16
YFIX	60	10	2	19	1008	1020	66 67	31 24	10.04
		25%	2	19	1008	1020	66 67	31 24	10.32
		5%	1	17	1048	980	75 57	35 -	10.28
V2YFIX	60	10%	1	17	1048	980	75 57	35 -	10.23
		25%	1	17	1048	980	75 57	35 -	10.33
		5%	1	17	1048	980	75 57	35 -	10.11
PROYFIX	60	10%	1	17	1048	980	75 57	35 -	10.33
		25%	1	17	1048	980	75 57	35 -	10.26

continued

Table 6.6 (continued)

L=2	M = 40		Measures of Goodness					CPU
	C_k	S_k	1	2	3	4	5	(sec)
		5%	0	0	982 1046	62 86	-	10.30
YFIX	100	10%	0	0	1182 846	67 79	-	10.27
		25%	0	0	1310 718	70 74	-	10.56
		5%	0	0	982 1046	62 86	-	10.30
V2YFIX	100	10%	0	0	1182 846	67 79	-	10.46
		25%	0	0	1310 718	70 74	-	10.36
		5%	0	0	982 1046	62 86	-	10.34
PROYFIX	100	10%	0	0	1182 846	67 79	-	10.28
		25%	0	0	1310 718	70 74	-	10.50

'dedicated,' which is a moot point. A block in column 5 with a combination of numerical entries and hash marks indicates that some of the available sequencers make only one production run, while other sequencers make two or more. The CPU time column is self-explanatory.

Several observations based on the numerical results in these tables are addressed. First, all of the CPU times are extremely reasonable. Even when the largest typical problems are run (M = 40), the CPU time does not exceed 11 seconds. Another observation is that many of the production schedules are identical for a given problem, regardless of which heuristic procedure is used. This is easily distinguishable by observing that all of the numerical entries are the same for a given problem over all three procedures.

Two of the problems result in production schedules that do not require any change-overs, regardless of the computed volume capacity, when the dispensing head limit (C_k) is set at 60 (Tables 6.1 and 6.3). Table 6.2, however, illustrates a situation where the computed volume capacity greatly affects the production schedule. Notice that the three procedures are run twice with a restricted volume capacity constant (5%). When 10% and 25% are inserted, 60 dispensing heads are sufficient to produce all of the assigned pack types without requiring a change-over.

Even with 100 dispensing heads, this particular problem required one sequencer change-over with the volume capacity constant set at 5%. This type of result indicates that managerial policy, with respect to efficient production operations, ought to consider the distribution of total workload as a contributing factor, possibly more so than an increase in the number of dispensing heads.

The final observation pertains to the relationship, or lack thereof, between the volume capacity constants and the balance of workload distribution across the sequencers. The great majority of the resulting production schedules reflect that the distribution of workload remains relatively balanced, regardless of the volume capacity constant (up to 25%). Obviously, a higher volume capacity constant could affect the outcome.

6.2 Specifically Difficult Problems in a Sequencing Environment

In order to exercise the algorithm completely, numerous large problems were concocted. Pack types were selected for these problems based on their large number of total different component types (b_i) required for production. The logic supporting this plan is that it seems reasonable to assume that pack types with large b_i values will tend to be more diversified with respect to their component compositions.

Tables 6.7 and 6.8 contain the numerical results corresponding to two selected problems. These two problems are quite typical of all problems we have solved in this category, and the results obtained are quite similar.

Table 6.7 presents a large problem consisting of 35 pack types. The numerical results in this table reflect only those production schedules devised with a volume capacity constant (S_k) of 10%. This is the most typical of the three constants used, and it illustrates the problem adequately. Each procedure solved this particular problem with dispensing head limits set at both 60 and 100. In addition, each procedure solved this problem with 2 and 3 available sequencers.

Table 6.7 Problem 7

M = 35				Measures of Goodness							CPU
		C_k	S_k	1	2	3			4	5	(sec)
YFIX	L=2	60	10%	9	226	4032	3428		96	23	7.14
									145	0	
		100	10%	2	66	4026	3434		98	-	7.84
									148	10	
	L=3	60	10%	8	205	2608	2760	2092	95	16	6.99
									132	1	
									84	21	
		100	10%	1	30	3166	1832	2462	85	-	7.43
									100	-	
									122	40	
V2YFIX	L=2	60	10%	9	238	3480	3980		111	15	6.96
									148	0	
		100	10%	2	60	4026	3434		98	-	7.67
									148	12	
	L=3	60	10%	8	216	2318	2284	2858	92	13	6.94
									131	2	
									114	4	
		100	10%	1	28	3166	1466	2828	85	-	7.41
									96	-	
									126	45	

continued

Table 6.7 (continued)

M = 35				Measures of Goodness					CPU
		C_{k}.	S_k	1	2	3	4	5	(sec)
	L=2	60	10%	9	227	3484 3976	107 148	16 0	7.01
PROYFIX		100	10%	2	60	4026 3434	98 148	- 12	7.60
	L=3	60	10%	8	196	2586 2420 2454	91 137 81	24 1 18	6.89
		100	10%	1	45	3166 2788 1506	85 141 87	- 45 -	7.50

Table 6.8 Problem 8

M = 40			Measures of Goodness						CPU
	C_k	S_k	1	2	3		4	5	(sec)
		5%	3	94	5186	4342	114	53	10.36
							152	25	
YFIX	100	10%	3	94	5186	4342	114	53	9.98
							152	25	
L=2		25%	3	94	5186	4342	114	53	9.98
							152	25	
		5%	3	105	5030	4498	130	61	10.15
							149	23	
V2YFIX	100	10%	3	105	5030	4498	130	61	10.14
							149	23	
L=2		25%	3	105	5030	4498	130	61	10.32
							149	23	
		5%	3	105	5030	4498	130	61	10.11
							149	23	
PROYFIX	100	10%	3	105	5030	4498	130	61	10.14
							149	23	
L=2		25%	3	105	5030	4498	130	61	9.72
							149	23	

continued

Table 6.8 (continued)

M = 40			Measures of Goodness							CPU
	C_k	S_k	1	2	3			4	5	(sec)
								89	-	
		5%	2	61	3594	3308	2626	152	8	10.03
								99	-	
								93	-	
YFIX	100	10%	2	48	3960	2462	3106	136	63	9.99
								106	28	
								99	-	
L=3		25%	2	66	4222	3336	1970	140	29	10.09
								96	-	
								89	-	
		5%	2	70	3594	2336	3598	133	56	10.05
								132	53	
								93	-	
V2YFIX	100	10%	2	67	3960	2336	3232	133	56	9.88
								129	54	
								99	-	
L=3		25%	2	70	4222	2174	3132	131	55	10.39
								121	52	
								89	-	
		5%	2	71	3594	3944	1990	140	55	9.82
								121	58	
								93	-	
PROYFIX	100	10%	2	67	3960	2336	3232	133	56	10.01
								129	54	
								99	-	
L=3		25%	2	80	4222	2512	2794	140	51	10.12
								129	49	

The interesting aspect of this problem is the fact that a strict increase in the number of sequencers does not necessarily result in a much more efficient production schedule. A large reduction in the total number of change-overs (column 1) occurs when the number of heads on the currently available sequencers increases. Merely introducing another sequencer into the situation does not appear to reduce the number of change-overs substantially.

The problem represented in Table 6.8 enforces the issue broached by the problem in Table 6.7. This particular problem serves to reveal that every production schedule devised for this problem has approximately the same total number of change-overs because the number of dispensing heads for each sequencer is fixed at 100. Regardless of the computed volume capacity alterations (S_k = 5%, 10% and 25%), or the number of sequencers available for production (L = 2 and 3), the number of change-overs only reduces from three to two. This small reduction is in stark contrast to the substantial reduction brought about by increasing the number of dispensing heads on the currently available sequencers in Table 6.7.

6.3 Conclusions

As mentioned earlier in this chapter, the purpose of this heuristic approach is to provide some sort of reasonable solution to the sequencer scheduling/assignment problem. This heuristic approach attempts to make sense out of a difficult problem (in this case, a variation of the sequenced tape production scheduling problem that is not well defined).

It appears that this approach does provide some structure to the problem. Indeed, given a particular problem from the set of simulated data, production schedules are devised that attempt to make the total amount of change-over time small and maintain a relatively balanced workload distribution. In this respect, the algorithm performs very well.

The algorithm does schedule all pack types to be produced for a particular problem, and it does so in a reasonable amount of CPU time. It is easily envisioned that a production supervisor could utilize these procedures prior to a planning period, and, in a short time span, determine a production schedule in order to produce all pack types in a reasonally efficient manner.

The algorithm performed well on large sets of data. Recall from Chapter 2 that the Fathi/Taheri IP model required roughly 46 minutes to solve an incredibly small problem. Overall, the heuristic algorithm does do what it is intended to do.

A final conclusion relates to the three different heuristic procedures. As evidenced by the numerical results in the tables, production schedules devised by the three procedures for the same problem vary little, if at all. This indicates that the different definitions of the term common may not be so different from each other. This fact is not alarming in that there are only so many ways by which to compare pack types to each other.

6.4 Avenues for Further Research

It is clear that the sequenced tape production scheduling problem is combinatorial in nature and, as such, is riddled with nuances that are not well explained. A vital key for success to more efficiently confront this problem is to achieve a better definition of the problem. Until a better understanding of this particular problem is ascertained, individuals faced with this problem are at a decided disadvantage.

With respect to the heuristic approach developed for this problem, a further investigation into this type of approach may result in even better strategies. Because of the fact that a heuristic solution cannot be compared to an 'optimal' one, the relative merits of a heuristic approach are difficult to determine.

Other measures of goodness can be identified in order to develop more reliable heuristic procedures. This area stems back to the not particularly well-defined problem of realizing when an optimal solution is achieved. In this case, we don't know how close we are, or how close we can get. The single most intriguing aspect of the sequenced tape production scheduling problem is that its structure is not readily apparent. When the exact structure of this problem is understood, a smoother, more straightforward path towards achieving an optimal solution will be much more accessible. A practical approach for accomplishing this is to conduct many more numerical experiments. In this manner, the mechanisms which cause the algorithm to perform in a certain way may be intensely scrutinized. Different strategies which might favorably impact on the efficiency of the algorithm could also be investigated.

CHAPTER 7

REFERENCES

1. Crowder, H., Johnson, E. L., and Padberg, M. (1983). "Solving Large-Scale Zero-One Linear Programming Problems," Operations Research, 31, No. 5, 803-834.

2. Fathi, Y., and Taheri, J. (1986). "A Mathematical Model for Loading Sequencers," Technical Report #86-17, North Carolina State University, Raleigh, NC.

3. Linear, Integer and Quadratic Programming with LINDO. (1986). Linus Schrage, University of Chicago, The Scientific Press.

4. Linear, Integer and Quadratic Programming with LINDO, User's Manual. (1984). Linus Schrage, University of Chicago, The Scientific Press.

5. McGinnis, Leon F. (1986). Personal communication. Georgia Institute of Technology, Atlanta, GA.

6. Murty, Katta G. (1983). Linear Programming, John Wiley & Sons, New York.

7. Ozan, Turgut M. (1986). Applied Mathematical Programming for Production and Engineering Management, Prentice-Hall, New Jersey.

8. Papadimitriou, Christos H., and Steiglitz, Kenneth. (1982). Combinatorial Optimization, Algorithms and Complexity, Prentice-Hall, New Jersey.

9. Saboo, S., Wang, L., and Wilhelm, W. E. (1986). "Development and Applications of Models to Manage Material Flow in Printed Circuit Board Assembly," Working Paper No. 1986-009, Dept. of Industrial and Systems Engineering, The Ohio State University.

10. Spielberg, K. (1979). "Enumerative Methods in Integer Programming," Ann. Discrete Math, 5, 139-183.

11. Salkin, Harvey M. (1975). Integer Programming, Addison-Wesley, Massachusetts.

12. Universal Instruments Corporation. (1986). Reference Manual for the Sequencing Machine, User's Manual, Kirkwood, New York.

CHAPTER 8

APPENDICES

(PROGRAM LISTINGS)

```
C  APPENDIX 8.1  MATRIX GENERATOR FOR LINDO
C
C **********************************************************************
C
C      THIS MATRIX GENERATOR SUBROUTINE FORMULATES THE FATHI/TAHERI
C INTEGER PROGRAMMING MODEL OF THE SEQUENCER ASSIGNMENT PROBLEM.
C THIS SUBROUTINE NAME IS 'USER'.
C
C **********************************************************************
C
C VARIABLE DEFINITION:
C
C VNAME- 8 ELEMENT INTEGER ARRAY CARRYING DESCRIPTION OF VARIABLES.
C IVAR- INTEGER VARIABLE THAT KEEPS COUNT OF THE NUMBER OF DECISION
C       VARIABLES GENERATED BY THE FORMULATION.
C NONZ- INTEGER VARIABLE DENOTING THE NUMBER OF TIMES A PARTICULAR
C       DECISION VARIABLE WILL APPEAR IN A FORMULATION.
C VAL- 42 ELEMENT REAL ARRAY CARRYING ALL COEFFICIENTS OF ANY
C      PARTICULAR VARIABLE.
C IRO- 42 ELEMENT INTEGER ARRAY CARRYING ALL OF THE ROWS IN WHICH ANY
C      PARTICULAR VARIABLE WILL APPEAR IN THE FORMULATION. ARRAYS
C      VAL AND IRO CORRESPOND SO THAT THE COEFFICIENT OF VARIABLE
C      Xjk STORED IN VAL(1) WILL APPEAR IN ROW IRO(1).
C ALFANM- 36 ELEMENT INTEGER ARRAY CARRYING THE DIGITS 0 THRU 9 AND
C         THE 26 LETTERS OF THE ALPHABET.
C A- AN M*N MATRIX ASSOCIATING COMPONENT TYPE j TO PACK TYPE i.
C    A '1' DENOTES THAT COMPONENT TYPE j IS REQUIRED ON PACK TYPE i;
C    A '0' DENOTES THAT IT IS NOT REQUIRED. i = 1...M; j = 1...N.
C B- A 40 ELEMENT INTEGER ARRAY STORING THE TOTAL NUMBER OF DIFFERENT
C    COMPONENT TYPES REQUIRED TO PRODUCE PACK TYPE i. EACH ARRAY
C    ELEMENT STORES THE TOTAL FOR ONE SPECIFIC PACK TYPE i.
C    Bi = SUMMATION Aij ; (Aij = 0 or 1).
C BPRIME- A 40 ELEMENT INTEGER ARRAY STORING THE TOTAL NUMBER OF
C         COMPONENTS REQUIRED TO PRODUCE PACK TYPE i.
C         FOR THESE TEST PROBLEMS, BPRIMEi = Bi + 5.
C V- A 40 ELEMENT INTEGER ARRAY STORING THE TOTAL NUMBER OF EACH
C    PACK TYPE i REQUIRED TO BE PRODUCED.
C VOLUME- A 40 ELEMENT INTEGER ARRAY STORING THE TOTAL NUMBER OF
C         COMPONENTS REQUIRED TO PRODUCE ALL OF EACH PACK TYPE i.
C         (VOLUMEi = BPRIMEi * Vi).
C C- A 6 ELEMENT REAL ARRAY STORING THE NUMBER OF AVAILABLE HEADS
C    ON SEQUENCER k; k = 1...L.
C S- A 6 ELEMENT REAL ARRAY STORING THE TOTAL NUMBER OF COMPONENTS
C    THAT SEQUENCER k IS PERMITTED TO INSERT TO PRODUCE ALL OF ITS
C    ASSIGNED SEQUENCED TAPES.
C L- INTEGER DENOTING THE NUMBER OF AVAILABLE SEQUENCERS.
C M- INTEGER DENOTING THE NUMBER OF PACK TYPES REQUIRED
C     TO BE PRODUCED.
C N- INTEGER DENOTING THE NUMBER OF COMPONENT TYPES REQUIRED TO
C    PRODUCE ALL PACK TYPES.
```

```
C MAXL- INTEGER VARIABLE DENOTING THE MAXIMUN NUMBER OF
C       SEQUENCERS ALLOWED.
C MAXM- INTEGER VARIABLE DENOTING THE MAXIMUM NUMBER OF
C       PACK TYPES ALLOWED.
C MAXN- INTEGER VARIABLE DENOTING THE MAXIMUM NUMBER OF
C       COMPONENT TYPES ALLOWED.
C BLANK- INTEGER VARIABLE DENOTING A BLANK SPACE.
C I1,I2,I3- 3 INTEGER VARIABLES USED TO DESCRIBE WHICH DECISION
C             VARIABLE IS BEING CHARACTERIZED AT ANY GIVEN TIME.
C ITEMP, JTEMP, KTEMP- 3 INTEGER VARIABLES USED FOR TEMPORARY
C                        STORAGE OF VALUES.
C IM- LOOP CONTROL VARIABLE USED WHEN LOOPING OVER PACK TYPES (M).
C IN- LOOP CONTROL VARIABLE USED WHEN LOOPING OVER COMPONENT TYPES (N).
C IL- LOOP CONTROL VARIABLE USED WHEN LOOPING OVER SEQUENCERS (L).
C COMCON- (COMPONENT CONSTANT): AN INTEGER CONSTANT ADDED TO ALL Bi
C          TO COMPUTE ALL BPRIMEi. (BPRIMEi = Bi + COMCON).
C LDCON- (LOAD CONSTANT): AN INTEGER CONSTANT USED TO COMPUTE ALL Sk.
C         Sk = (SUMMATION i=1...M, (Vi*BPRIMEi)/L + LDCON).
C TRUBLE- LOGICAL VARIABLE; TRUBLE IS RETURNED .TRUE. IF A PROBLEM
C           OCCURS; ie. IF AN OUT-OF-SPACE CONDITION EXISTS WHEN
C           LINDO SUBROUTINES 'DEFROW' OR 'APPCOL' ARE CALLED.
C
C *******************************************************************
C
      SUBROUTINE USER
C
C *******************************************************************
C
C DECLARATION OF VARIABLES:
C
DIMENSION VNAME(8), VAL(42), IRO(42), ALFANM(36), VOLUME(40)
DIMENSION A(40,400), B(40), BPRIME(40), V(40), S(6), C(6)
LOGICAL TRUBLE
INTEGER VNAME, IRO, ALFANM, BLANK, NONZ, A, B, BPRIME, VOLUME
INTEGER V, L, M, N, IM, IN, IL, COMCON, LDCON, IVAR
INTEGER I1, I2, I3, ITEMP, JTEMP, KTEMP, MAXL, MAXM, MAXN
      REAL C, S, VAL
C
C *******************************************************************
C
C DEFINE 'BLANK' AND ARRAY 'ALFANM'
C
DATA BLANK/' '/
DATA ALFANM/'0','1','2','3','4','5','6','7','8','9',
     +            'A','B','C','D','E','F','G','H','I','J','K','L','M',
     +            'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'/
C
C INITIALIZE 'COMCON' AND SET UPPER BOUNDS FOR NUMBER OF
C   SEQUENCERS (L), NUMBER OF PACK TYPES (M), AND NUMBER OF COMPONENT
C   TYPES (N).
C
```

```
CONCON = 5
C MAX NUMBER OF SEQUENCERS (L)
MAXL = 6
C MAX NUMBER OF PACK TYPES (M)
MAXM = 40
C MAX NUMBER OF COMPONENT TYPES (N)
MAXN = 400
C
C ***********************************************************************
C
C READ IN KNOWN DATA
C
C  OPEN DATA FILE
OPEN(UNIT=7,FILE='DATA.DAT',STATUS='OLD')
C
C READ NUMBER OF SEQUENCERS(L), NUMBER OF PACK TYPES(M), AND NUMBER
C   OF COMPONENT TYPES(N).
C
READ(7,*) L,M,N
C CHECK TO SEE THAT L,M, AND N DO NOT EXCEED UPPER BOUNDS
C
IF (L .GT. MAXL) WRITE (*,1000) MAXL
IF (M .GT. MAXM) WRITE (*,1001) MAXM
IF (N .GT. MAXN) WRITE (*,1002) MAXN
C
1000 FORMAT (/,' PROGRAM TERMINATED - NUMBER OF SEQUENCERS (L)
     +  EXCEEDS MAX ALLOWED:', I4)
1001 FORMAT (/,' PROGRAM TERMINATED - NUMBER OF PACK TYPES (M)
     +  EXCEEDS MAX ALLOWED:', I5)
1002 FORMAT (/,' PROGRAM TERMINATED - NUMBER OF COMPONENT TYPES (N)
     +  EXCEEDS MAX ALLOWED:', I6)
IF((L .GT. MAXL) .OR. (M .GT. MAXM) .OR. (N .GT. MAXN))GO TO 99
C
C READ NUMBER OF EACH PACK TYPE TO BE PRODUCED(Vi).
READ(7,*) (V(IM), IM = 1,M)
C READ NUMBER OF HEADS AVAILABLE ON EACH SEQUENCER(Ck).
READ(7,*) (C(IL), IL = 1,L)
C READ MATRIX 'A'
DO 5 IM = 1,M
   READ(7,*) (A(IM,IN), IN = 1,N)
5 CONTINUE
C READ LDCON
READ (7,*) LDCON
C
C  CLOSE DATA FILE
CLOSE(UNIT=7)
C
C ALL KNOWN DATA IS ENTERED
C
C ***********************************************************************
C
```

```
C COMPUTE ALL Bi, BPRIMEi, VOLUMEi, AND Sk AND STORE RESULTS IN
C   ARRAYS 'B', 'BPRIME', 'VOLUME' AND 'S' RESPECTIVELY.
C
DO 10 IM =1,M
   B(IM) = 0
   DO 15 IN = 1,N
      B(IM) = B(IM) + A(IM,IN)
15    CONTINUE
   BPRIME(IM) = B(IM) + COMCON
10 CONTINUE
C
DO 20 IL = 1,L
   S(IL) = 0.
   DO 25 IM = 1,M
      VOLUME(IM) = V(IM)*BPRIME(IM)
      S(IL) = S(IL) + FLOAT(VOLUME(IM))
25    CONTINUE
         S(IL) = S(IL)/FLOAT(L) + FLOAT(LDCON)
20 CONTINUE
C
C ARRAYS 'B', 'BPRIME', 'VOLUME', AND 'S' ARE FILLED
C
C ******************************************************************
C
C PRINT L,M,N AND LDCON AND ARRAYS 'B', 'BPRIME', 'V', 'VOLUME',
C    'S', AND 'C'
C
C  OPEN OUTPUT FILE
OPEN(UNIT=15,FILE='PARAM.DAT',STATUS='NEW')
C
WRITE (15,2000) L,M,N,LDCON
WRITE (15,2001) (B(IM), IM = 1,M)
WRITE (15,2002) (BPRIME(IM), IM = 1,M)
WRITE (15,2003) (V(IM), IM = 1,M)
WRITE (15,2004) (VOLUME(IM), IM = 1,M)
WRITE (15,2005) (S(IL), IL = 1,L)
WRITE (15,2006) (C(IL), IL = 1,L)
C
2000  FORMAT ('0', 'L=',I2,3X,'M=',I3,3X,'N=',I4,3X,'LDCON=',I7)
2001 FORMAT ('0', 'ARRAY B:', 15(1X,I3))
2002 FORMAT ('0', 'ARRAY BPRIME:', 10(1X,I5))
2003 FORMAT ('0', 'ARRAY V:', 15(1X,I3))
2004 FORMAT ('0', 'ARRAY VOLUME:', 10(1X,I5))
2005 FORMAT ('0', 'ARRAY S:', 6(1X,F10.2))
2006 FORMAT ('0', 'ARRAY C:', 6(1X,F5.1))
C
C  CLOSE OUTPUT FILE
C CLOSE(UNIT=15)
C
C ******************************************************************
C
```

```
C INITIALIZE THE ROWS OF THE FORMULATION
C
C ********************************************************************
C
CALL INIT
C
C OBJECTIVE ROW
CALL DEFROW(1,0.,IDROW,TRUBLE)
C
C PACK-TYPE/SEQUENCER CONSTRAINTS
DO 40 IM = 1,M
   CALL DEFROW(0,1.,IDROW,TRUBLE)
40 CONTINUE
C
C LOAD BALANCING CONSTRAINTS
DO 50 IL = 1,L
   CALL DEFROW(1,S(IL),IDROW,TRUBLE)
50 CONTINUE
C
C HEAD UTILIZATION CONSTRAINTS
DO 60 IL = 1,L
   CALL DEFROW(-1,C(IL),IDROW,TRUBLE)
60 CONTINUE
C
C COMPONENT/SEQUENCER CONSTRAINTS
DO 70 IM = 1,M
   DO 80 IL = 1,L
      CALL DEFROW(-1,0.,IDROW,TRUBLE)
80    CONTINUE
70 CONTINUE
C
C ROW DEFINITION COMPLETE
C
C ********************************************************************
C VARIABLE DESCRIPTION - GENERATE THE 'Y' DECISION VARIABLES
C
C ********************************************************************
C
C PREPARE ARRAY 'VNAME' FOR THE 'Y' VARIABLES
C
VNAME(1) = ALFANM(35)
VNAME(5) = BLANK
VNAME(6) = BLANK
VNAME(7) = BLANK
VNAME(8) = BLANK
IVAR = 0
NONZ = 4
C
C DETERMINE THE 'i' SUBSCRIPT OF 'Yik' BY SETTING I1 AND I2 = 2 DIGITS
C   OF THE PACK TYPE (i = 1...M).
DO 100 IM = 1,M
```

```
   I1 = IM/10
   I2 = IM-I1*10
C
C PLACE I1,I2 IN THE VARIABLE NAME
   VNAME(2) = ALFANM(I1+1)
   VNAME(3) = ALFANM(I2+1)
C
C DETERMINE THE 'k' SUBSCRIPT OF 'Yik'
   DO 200 IL = 1,L
      VNAME(4) = ALFANM(IL+10)
C
C OBJECTIVE COEFFICIENT
              VAL(1) = 0.
      IRO(1) = 1
C
C PACK-TYPE/SEQUENCER CONSTRAINTS
      VAL(2) = 1.
      IRO(2) = 1+IM
C
C LOAD BALANCING CONSTRAINTS
      VAL(3) = FLOAT(V(IM)*BPRIME(IM))
      IRO(3) = 1+M+IL
C
C COMPONENT/SEQUENCER CONSTRAINTS
      VAL(4) = -FLOAT(B(IM))
      IRO(4) = 1+M+2*L+L*(IM-1)+IL
C
C PLACE THE 'Y' VARIABLES IN THEIR CORRECT ROWS
C
      CALL APPCOL(VNAME,NONZ,VAL,IRO,TRUBLE)
              IVAR = IVAR + 1
C
C SET AN UPPER BOUND OF '1' FOR ALL 'Y' VARIABLES
C
              CALL SETSUB(IVAR,1.)
C
200   CONTINUE
100 CONTINUE
C
C GENERATION OF ALL 'Y' DECISION VARIABLES IS COMPLETE
C
C **********************************************************************
C
C GENERATE THE 'X' DECISION VARIABLES
C
C **********************************************************************
C
C PREPARE ARRAY 'VNAME' FOR THE 'X' VARIABLES
C
VNAME(1) = ALFANM(34)
NONZ = M+2
```

```
C
C DETERMINE THE 'j' SUBSCRIPT OF 'Xjk' BY SETTING I1,I2, AND
C   I3 = 3 DIGITS OF THE COMPONENT TYPE (j = 1...N).
DO 300 IN = 1,N
   I1 = IN/100
   ITEMP = IN-I1*100
   I2 = ITEMP/10
   I3 = ITEMP-I2*10
C
C PLACE I1,I2,AND I3 IN THE VARIABLE NAME
   VNAME(2) = ALFANM(I1+1)
   VNAME(3) = ALFANM(I2+1)
   VNAME(4) = ALFANM(I3+1)
C
C DETERMINE THE 'k' SUBSCRIPT OF 'Xjk'
   DO 400 IL = 1,L
      VNAME(5) = ALFANM(IL+10)
C
C OBJECTIVE COEFFICIENT
      VAL(1) = 1.
      IRO(1) = 1
C
C HEAD UTILIZATION CONSTRAINTS
      VAL(2) = 1.
      IRO(2) = 1+M+L+IL
C
C COMPONENT/SEQUENCER CONSTRAINTS
      JTEMP = M+2
      DO 500 IM = 3,JTEMP
         KTEMP = IM-2
 VAL(IM) = FLOAT(A(KTEMP,IN))
 IRO(IM) = 1+M+2*L+L*(IM-3)+IL
500      CONTINUE
C
C PLACE THE 'X' VARIABLES IN THEIR CORRECT ROWS
C
   CALL APPCOL(VNAME,NONZ,VAL,IRO,TRUBLE)
   IVAR = IVAR + 1
C
C SET AN UPPER BOUND OF '1' FOR ALL 'X' VARIABLES
C
          CALL SETSUB(IVAR,1.)
C
400    CONTINUE
300 CONTINUE
C
C GENERATION OF ALL 'X' DECISION VARIABLES IS COMPLETE
C THE MODEL FORMULATION IS COMPLETE. END OF SUBROUTINE 'USER'.
C
RETURN
99 END
```

```
C  APPENDIX 8.2  YFIX HEURISTIC PROCEDURE
C
C ********************************************************************
C ********************************************************************
C
C      THIS 'YFIX' HEURISTIC PROCEDURE DEVISES A PRODUCTION SCHEDULE
C OF SEQUENCED TAPES FOR THE SEQUENCER SCHEDULING/ASSIGNMENT PROBLEM.
C
C ********************************************************************
C ********************************************************************
C
C VARIABLE DEFINITION:
C ********************************************************************
C
C    THE FOLLOWING VARIABLES ARE TYPE 'INTEGER' --
C
C VOLUME(  ) : CONTAINS TOTAL# OF COMPONENTS REQUIRED TO PRODUCE ALL
C               UNITS OF EACH PACK TYPE.
C A(  ,  ) : THIS IS MATRIX 'A' WHICH SHOWS WHICH COMPONENT TYPES ARE
C             REQUIRED BY EACH PACK TYPE.
C B(  ) : CONTAINS THE TOTAL# OF DIFFERENT COMPONENT TYPES REQUIRED
C         BY EACH PACK TYPE.
C BPRIME(  ,  ) : CONTAINS THE TOTAL# OF EACH COMPONENT TYPE REQUIRED
C                  BY EACH PACK TYPE.
C HEADS(  ) : CONTAINS THE NUMBER OF DISPENSING HEADS ON EACH SEQUENCER.
C L : INTEGER DENOTING THE NUMBER OF AVAILABLE SEQUENCERS.
C M : INTEGER DENOTING THE NUMBER OF PACK TYPES TO BE PRODUCED.
C N : INTEGER DENOTING THE NUMBER OF COMPONENT TYPES.
C IM,IN,IL,JM : LOOP CONTROL VARIABLES.
C DIFFER(  ,  ) : (M+L)*M MATRIX REPRESENTING THE DEGREE OF SIMILARITY
C                  BETWEEN PACK TYPES.
C FIXED(  ) : ARRAY USED TO FILL THE FIRST 'M' ROWS OF ARRAY 'DIFFER'.
C COMP(  ) : ARRAY USED TO COMPARE PACK TYPES.
C SMALL : INTEGER REPRESENTING A NUMERICAL ENTRY IN ARRAY 'DIFFER'.
C INDEX : INTEGER REPRESENTING A SELECTED OR ASSIGNED PACK TYPE.
C REMAIN(  ) : CONTAINS THE NUMBER OF UNALLOCATED HEADS ON SEQUENCERS.
C ALCATE(  ,  ) : INDICATES THE ASSIGNMENT OF PACK TYPES TO SEQUENCERS.
C V(  ) : CONTAINS THE NUMBER OF EACH PACK TYPE TO BE PRODUCED.
C UNION(  ) : THIS IS THE 'U' VECTOR INDICATING THE AGGREGATION OF
C              DIFFERENT COMPONENT TYPES REQUIRED BY A SEQUENCER
C              FOR A PARTICULAR PRODUCTION RUN.
C ITEMP : TEMPORARY VARIABLE USED TO DUPLICATE 'INDEX'.
C NEXT : INDICATES THE APPROPRIATE M+kth ROW OF ARRAY 'DIFFER'.
C LOAD(  ) : CONTAINS THE CURRENT PRODUCTION VOLUME REQUIREMENT
C             OF EACH SEQUENCER.
C SUM : INTEGER DENOTING THE CURRENT NUMBER OF ASSIGNED PACK TYPES.
C LIGHT : REPRESENTS THE SEQUENCER WITH THE SMALLEST CUMULATIVE
C         TOTAL PRODUCTION VOLUME REQUIREMENT IN PART TWO.
C KEY : REPRESENTS THE SEQUENCER BEING CURRENTLY SCHEDULED IN PART TWO.
C MAX : THE TOTAL# OF ROWS IN ARRAY 'DIFFER'.
C TOTAL(  ) : CONTAINS TOTAL# OF COMPONENTS REQUIRED TO PRODUCE ONE
```

```
C              UNIT OF EACH PACK TYPE.
C ORDER( ) : CONTAINS THE PACK TYPES, IN ASCENDING ORDER, WHICH ARE
C             DESIGNATED TO BE PRODUCED.
C BOARD : REPRESENTS AN INDIVIDUAL PACK TYPE.
C COMPNT : REPRESENTS AN INDIVIDUAL COMPONENT TYPE.
C NUM : THE NUMBER OF EACH COMPONENT TYPE REQUIRED BY AN INDIVIDUAL
C       PACK TYPE.
C OHAUL( ) : CONTAINS THE TOTAL# OF CHANGE-OVERS REQUIRED BY
C             EACH SEQUENCER.
C TRACK( , , ) : INDICATES THE PACK TYPES ASSIGNED TO EACH SEQUENCER
C                 FOR EACH PRODUCTION RUN.
C MAXRUN : THE TOTAL# OF PRODUCTION RUNS MADE BY A SEQUENCER.
C HOLD( ) : REPRESENTS THE DEGREE OF COMMONALITY BETWEEN EACH
C            UNASSIGNED PACK TYPE, AND THE AGGREGATION OF PREVIOUSLY
C            ASSIGNED PACK TYPES.
C MOST : INDICATES THE NUMBER OF COMPONENT TYPES REQUIRED BY EACH
C        UNASSIGNED PACK TYPE AND THE AGGREGATION OF PREVIOUSLY
C        ASSIGNED PACK TYPES ON A PARTICULAR SEQUENCER FOR A
C        PARTICULAR PRODUCTION RUN.
C VOLTOT : THE TOTAL# OF COMPONENT INSERTIONS REQUIRED TO PRODUCE
C          ALL OF THE ASSIGNED PACK TYPES.
C EQUAL : THE VOLUME CAPACITY OF EACH SEQUENCER'S FIRST PRODUCTION RUN.
C COTYPE( , , ) : INDICATES THE COMPONENT TYPES REQUIRED BY EACH
C                  SEQUENCER FOR EACH PRODUCTION RUN.
C COUNT : INTEGER VARIABLE USED TO KEEP COUNT.
C TYPE( ) : CONTAINS THE COMPONENT TYPES REQUIRED BY EACH SEQUENCER
C            FOR EACH PRODUCTION RUN.
C SPACES( , ) : CONTAINS THE NUMBER OF COMPONENT TYPES REQUIRED BY
C                EACH SEQUENCER FOR EACH PRODUCTION RUN.
C LDCON : THE VOLUME CAPACITY CONSTANT USED TO CALCULATE THE
C         SEQUENCER VOLUME CAPACITY IN PART ONE.
C SEQTOT( ) : CONTAINS THE TOTAL# OF COMPONENT TYPES REQUIRED BY
C              EACH SEQUENCER TO MAKE ALL OF IT'S PRODUCTION RUNS.
C OTHER( , ) : CONTAINS THE NUMBER OF COMPONENT TAPE CHANGES
C               REQUIRED BY EACH SEQUENCER BETWEEN CONSECUTIVE
C               PRODUCTION RUNS.
C INTSEC( ) : CONTAINS THE NUMBER OF DEDICATED DISPENSING HEADS
C              ON EACH SEQUENCER.
C INITIM : CPU TIMER VARIABLE.
C
C    THE FOLLOWING VARIABLES ARE TYPE 'LOGICAL' --
C
C FILLED : USED TO FILL THE FIRST 'M' ROWS OF ARRAY 'DIFFER'.
C PACK( ) : INDICATES WHEN A PACK TYPE IS ASSIGNED FOR PRODUCTION.
C PACK2( ) : INDICATES WHEN A PACK TYPE IS ASSIGNED FOR PRODUCTION
C             IN PART TWO OF THE ALGORITHM.
C
C *******************************************************************
C SUBROUTINE DEFINITION:
C *******************************************************************
C
```

```
C RESULT : SELECTS THE 'LEAST' OR 'MOST' COMMON UNASSIGNED PACK TYPE.
C
C ASSIGN : ASSIGNS A PACK TYPE TO A SEQUENCER, UPDATES ARRAY 'PACK',
C          UPDATES PVRk, INDICATES THE SEQUENCER AND PRODUCTION RUN
C          TO WHICH A PACK TYPE IS ASSIGNED, AND UPDATES THE TOTAL
C          NUMBER OF ASSIGNED PACK TYPES.
C
C NEXPAC : SELECTS THE MOST SIMILAR UNASSIGNED PACK TYPE.
C
C COMPARE : ESTABLISHES THE 'U' VECTOR.
C
C NEWDIF : UPDATES THE APPROPRIATE M+kth ROW OF ARRAY 'DIFFER'.
C
C **********************************************************************
C DECLARATION OF VARIABLES:
C **********************************************************************
C
INTEGER VOLUME(40), A(40,210), B(40), BPRIME(40,210), HEADS(6)
        INTEGER L, M, N, IM, IN, IL, JM, DIFFER(46,40), FIXED(210)
        INTEGER COMP(210), SMALL, INDEX, REMAIN(6), ALCATE(6,40), V(40)
        INTEGER UNION(210), ITEMP, NEXT, LOAD(6), SUM, LIGHT, KEY, MAX
        INTEGER TOTAL(40), ORDER(40), BOARD, COMPNT, NUM, OHAUL(6)
        INTEGER TRACK(6,10,40), MAXRUN, HOLD(40), MOST, VOLTOT, EQUAL
        INTEGER COTYPE(6,10,210), COUNT, TYPE(210), SPACES(6,10), LDCON
        INTEGER SEQTOT(6), OTHER(6,10), INTSEC(6), INITIM
LOGICAL FILLED, PACK(40), PACK2(40)
C
C **********************************************************************
C READ IN KNOWN DATA
C **********************************************************************
C
C SET SEQUENCER VOLUME CAPACITY CONSTANT (5%=20, 10%=10, 25%=4)
LDCON = 10
C
C OPEN DATA FILES
OPEN(UNIT=10,FILE='KNOWN.DAT',STATUS='OLD')
OPEN(UNIT=11,FILE='INPUT.DAT',STATUS='OLD')
OPEN(UNIT=12,FILE='SUBSET.DAT',STATUS='OLD')
C
C READ IN L,M,N, V(M), HEADS(L), ORDER(M)
C
READ(10,*) L,M,N
READ(10,*) (V(IM), IM = 1,M)
READ(10,*) (HEADS(IL), IL = 1,L)
READ(12,*) (ORDER(IM), IM = 1,M)
C
C READ IN THE TYPE (COMPNT), AND NUMBER OF EACH TYPE (NUM), OF
C   COMPONENT REQUIRED BY EACH PACK TYPE TO BE PRODUCED (BOARD)
C
4000 READ(11,1000,END=4200) BOARD, COMPNT, NUM
DO 5 IM = 1,M
```

```
   IF(BOARD .EQ. ORDER(IM)) GO TO 4100
   IF(IM .EQ. M .AND. BOARD .GT. ORDER(IM)) GO TO 4200
5 CONTINUE
GO TO 4000
C
C FILL MATRIX 'A' AND MATRIX 'BPRIME'
C
4100 A(IM,COMPNT) = 1
BPRIME(IM,COMPNT) = NUM
GO TO 4000
C
C CLOSE DATA FILES
4200 CLOSE(UNIT=10)
CLOSE(UNIT=11)
CLOSE(UNIT=12)
C
C INITIALIZE ARRAYS 'PACK' AND 'PACK2', AND COMPUTE ALL Bi, TOTALi,
C   VOLUMEi, THE TOTAL PRODUCTION VOLUME REQUIREMENT (VOLTOT), AND
C   THE SEQUENCER VOLUME CAPACITY PERMITTED IN THE FIRST RUN (EQUAL).
C
DO 10 IM = 1,M
C
   PACK(IM) = .TRUE.
   PACK2(IM) = .TRUE.
C
   B(IM) = 0
   TOTAL(IM) = 0
   DO 15 IN = 1,N
      B(IM) = B(IM) + A(IM,IN)
      TOTAL(IM) = TOTAL(IM) + BPRIME(IM,IN)
15    CONTINUE
C
C CHECK THAT THE NUMBER OF COMPONENT TYPES REQUIRED BY PACK(IM) IS
C   GREATER THAN ZERO AND NOT GREATER THAN 100. (ie. 0 < B(IM) <=  100)
C
   IF (B(IM) .EQ. 0 .OR. B(IM) .GT. 100) THEN
      WRITE(*,1012) ORDER(IM), B(IM)
      GO TO 9900
   ENDIF
10 CONTINUE
C
1012 FORMAT (/, ' PROGRAM TERMINATED - NUMBER OF COMPONENT TYPES
     + REQUIRED BY PACK',I4,' EQUALS',I4)
C
VOLTOT = 0
DO 20 IM = 1,M
   VOLUME(IM) = V(IM)*TOTAL(IM)
   VOLTOT = VOLTOT + VOLUME(IM)
20 CONTINUE
EQUAL = VOLTOT/L + VOLTOT/LDCON
C
```

```
C END OF DATA ENTRY AND DATA MANIPULATION
C
C START THE CPU TIMER
IF(.NOT. LIB$INIT_TIMER(INITIM)) GO TO 9900
C
C ********************************************************************
C FILL THE FIRST 'N' ROWS OF ARRAY 'DIFFER'
C ********************************************************************
C
DO 25 IM = 1,M
   FILLED = .FALSE.
   DO 30 JM = 1,M
      IF(JM .EQ. IM) THEN
 DIFFER(IM,JM) = 500
      ELSE
         DIFFER(IM,JM) = 0
 DO 35 IN = 1,N
    IF(.NOT. FILLED) FIXED(IN) = A(IM,IN)
    COMP(IN) = A(JM,IN)
    IF(FIXED(IN) .EQ. 0 .AND. COMP(IN) .EQ. 1) THEN
                DIFFER(IM,JM) = DIFFER(IM,JM) + 1
    ENDIF
35          CONTINUE
 FILLED = .TRUE.
              ENDIF
30          CONTINUE
25 CONTINUE
C
C ********************************************************************
C ********************************************************************
C
C PART ONE OF THE HEURISTIC ALGORITHM -
C   DETERMINES INITIAL ASSIGNMENT OF PACK TYPES FOR EACH SEQUENCER
C
C ********************************************************************
C ********************************************************************
C
C OPEN OUTPUT FILE
OPEN(UNIT=21,FILE='INFO.DAT',STATUS='NEW')
C
C INITIALIZE THE NUMBER OF PACKS ASSIGNED (SUM), THE MAXIMUM NUMBER OF
C   PRODUCTION RUNS ALLOWED (MAXRUN), THE STARTING PRODUCTION VOLUME
C   REQUIREMENT OF EACH SEQUENCER (LOAD), AND THE NUMBER OF CHANGE-
C   OVERS MADE BY EACH SEQUENCER.
C
SUM = 0
MAXRUN = 10
DO 50 IL = 1,L
   LOAD(IL) = 0
   OHAUL(IL) = 0
   DO 55 JM = 1,MAXRUN
```

```
      DO 60 IN = 1,N
 COTYPE(IL,JM,IN) = 0
60             CONTINUE
55    CONTINUE
50 CONTINUE
C
C SELECT THE FIRST PACK TYPE FOR ASSIGNMENT ON EACH SEQUENCER *********
C                 .
DO 100 IL = 1,L
   NEXT = M + IL
   IF(IL .EQ. 1) THEN
      SMALL = 500
   ELSE
      SMALL = 0
   ENDIF
C
C USE THE STARTING RULE FOR THE FIRST SEQUENCER **********************
C
   DO 105 IM =1,M
      IF(IL .EQ.1) THEN
         IF(B(IM) .LT. SMALL) THEN
            INDEX = IM
    ITEMP = INDEX
    SMALL = B(IM)
 ENDIF
C
C USE THE 'LEAST COMMON' RULE FOR THE LAST L - 1 SEQUENCERS ***********
C
      ELSE
C
C FILL ARRAY 'HOLD' WITH THE 'MOST COMMON' UNASSIGNED PACK TYPES ******
C
        IF(DIFFER(NEXT-1,IM) .LT. HOLD(IM))
     +                          HOLD(IM) =  DIFFER(NEXT-1,IM)
C
C SELECT THE 'LEAST COMMON' UNASSIGNED PACK TYPE *********************
C
 IF(HOLD(IM) .GT. SMALL .AND. PACK(IM)) THEN
    SMALL = HOLD(IM)
    CALL RESULT (INDEX,IM,ITEMP,SMALL,MOST,B(IM))
        ELSE
C
C IF A TIE EXISTS, IMPLEMENT THE HDOS PROCEDURE **********************
C
    IF(HOLD(IM) .EQ. SMALL .AND.
     +                        (B(IM) - SMALL) .LT. MOST) THEN
       CALL RESULT (INDEX,IM,ITEMP,SMALL,MOST,B(IM))
            ENDIF
C
                 ENDIF
              ENDIF
```

```
105       CONTINUE
   SMALL = B(INDEX)
C
C THE FIRST PACK TYPE IS SELECTED *************************************
C
C NOTE THE PACK TYPE SELECTED, AND IT'S CORRESPONDING Bi
C WRITE(21,*) ' '
C WRITE(21,1005) INDEX, SMALL, MOST
C
C ASSIGN THE PACK TYPE TO THE kth SEQUENCER AND UPDATE PVRk ***********
C
CALL ASSIGN(VOLUME(INDEX),ORDER(INDEX),LOAD(IL),SUM,
     +              ALCATE(IL,INDEX),PACK(INDEX),TRACK(IL,1,INDEX))
C
C CHECK IF ALL PACKS ARE ASSIGNED *************************************
IF(SUM .EQ. M) THEN
   DO 107 IN = 1,N
      IF(A(INDEX,IN) .EQ. 1)  COTYPE(IL,1,IN) = 1
107       CONTINUE
   GO TO 9000
ENDIF
C
C UPDATE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk) ********************
C
REMAIN(IL) = HEADS(IL) - SMALL
C
C NOTE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk)
C WRITE(21,1010) IL, REMAIN(IL)
C
C LOCATE THE 'MOST SIMILAR' UNASSIGNED PACK TYPE **********************
C
SMALL = 500
DO 110 IM = 1,M
C
   CALL NEXPAC(PACK(IM),DIFFER(ITEMP,IM),SMALL,INDEX,
     +                MOST,B(IM),IM)
C
110 CONTINUE
C
C NOTE THE PACK TYPE SELECTED, AND IT'S CORRESPONDING Bi
C WRITE(21,1005) INDEX, SMALL, MOST
C
C CHECK IF THIS UNASSIGNED PACK TYPE CAN BE ASSIGNED GIVEN THE
C   CURRENT Hk AND PVRk ***********************************************
C
IF((REMAIN(IL) - SMALL) .GE. 0 .AND.
     +        (LOAD(IL) + VOLUME(INDEX)) .LE. EQUAL) THEN
C
C ASSIGN THE PACK TYPE TO THE kth SEQUENCER AND UPDATE PVRk ***********
C
   CALL ASSIGN(VOLUME(INDEX),ORDER(INDEX),LOAD(IL),SUM,
     +                   ALCATE(IL,INDEX),PACK(INDEX),TRACK(IL,1,INDEX))
C
C UPDATE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk) ********************
C
           REMAIN(IL) = REMAIN(IL) - SMALL
C
C NOTE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk)
C WRITE(21,1010) IL, REMAIN(IL)
C
C INITIALIZE ARRAY 'UNION'
   DO 112 IN = 1,N
      UNION(IN) = 0
112    CONTINUE
C
C IF THIS UNASSIGNED PACK TYPE CANNOT BE ASSIGNED GIVEN THE CURRENT
C   Hk AND/OR PVRk, COPY THE APPROPRIATE ith ROW OF 'DIFFER' INTO
C   THE APPROPRIATE M+kth ROW, AND BEGIN SCHEDULING THE NEXT SEQUENCER.
C
ELSE
   DO 115 IM = 1,M
      DIFFER(NEXT,IM) = DIFFER(ITEMP,IM)
      IF(NEXT .EQ. (M+1))  HOLD(IM) = DIFFER(NEXT,IM)
115    CONTINUE
   DO 117 IN = 1,N
      COTYPE(IL,1,IN) = A(ITEMP,IN)
117    CONTINUE
C
           GO TO 100
ENDIF
C
C WHEN TWO OR MORE PACKS ARE ASSIGNED TO THE SAME SEQUENCER,
C   PERFORM THE COMPARISON OPERATION **********************************
C
5000 DO 120 IM = 1,M
   IF(ALCATE(IL,IM) .EQ. 1) THEN
              DO 125 IN = 1,N
C
         CALL COMPARE(A(IM,IN),UNION(IN),COTYPE(IL,1,IN))
C
125           CONTINUE
           ENDIF
120     CONTINUE
C
C CHECK IF ALL PACKS ARE ASSIGNED *************************************
IF(SUM .EQ. M) GO TO 9000
C
C NOTE THE 'U' VECTOR
C WRITE(21,*) 'ARRAY UNION:'
C WRITE(21,1003) (UNION(IN),IN = 1,N)
C
DO 130 IM = 1,M
```

```
   IF(PACK(IM)) THEN
      DIFFER(NEXT,IM) = 0
      DO 135 IN = 1,N
C
         CALL NEWDIF(COMP(IN),A(IM,IN),UNION(IN),
     +                        DIFFER(NEXT,IM))
C
135       CONTINUE
       ELSE
      DIFFER(NEXT,IM) = 500
   ENDIF
130     CONTINUE
C
C COMPARISON OPERATION IS COMPLETE ***********************************
C
C NOTE THE UPDATED M+kth ROW OF ARRAY 'DIFFER'
C WRITE(21,*) ' '
C WRITE(21,*) 'DIFFER:'
C WRITE(21,1002) (DIFFER(NEXT,IM), IM = 1,M)
C WRITE(21,*) ' '
C
C LOCATE THE 'MOST SIMILAR' UNASSIGNED PACK TYPE *********************
C
SMALL = 500
DO 140 IM = 1,M
C
   CALL NEXPAC(PACK(IM),DIFFER(NEXT,IM),SMALL,INDEX,
     +                  MOST,B(IM),IM)
C
140 CONTINUE
C
C NOTE THE PACK TYPE SELECTED, AND IT'S CORRESPONDING Bi
C WRITE(21,1005) INDEX, SMALL, MOST
C
C CHECK IF THIS UNASSIGNED PACK TYPE CAN BE ASSIGNED GIVEN THE
C   CURRENT Hk AND PVRk ***********************************************
C
IF((REMAIN(IL) - SMALL) .GE. 0 .AND.
     +         (LOAD(IL) + VOLUME(INDEX)) .LE. EQUAL) THEN
C
C ASSIGN THE PACK TYPE TO THE kth SEQUENCER AND UPDATE PVRk ***********
C
   CALL ASSIGN(VOLUME(INDEX),ORDER(INDEX),LOAD(IL),SUM,
     +                  ALCATE(IL,INDEX),PACK(INDEX),TRACK(IL,1,INDEX))
C
C UPDATE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk) *******************
C
          REMAIN(IL) = REMAIN(IL) - SMALL
C
C NOTE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk)
C WRITE(21,1010) IL, REMAIN(IL)
```

```
C
C PERFORM THE COMPARISON OPERATION. CONTINUE THIS LOOP UNTIL
C   THE kth SEQUENCER IS FULLY SCHEDULED ******************************
C
   GO TO 5000
ENDIF
C
C INITIALIZE ARRAY 'HOLD' AFTER THE FIRST SEQUENCER IS SCHEDULED ******
C
IF(NEXT .EQ. (M+1)) THEN
   DO 145 IM = 1,M
      HOLD(IM) = DIFFER(NEXT,IM)
145    CONTINUE
ENDIF
C
C NOTE ARRAY 'HOLD'
C WRITE(21,*) 'HOLD:'
C WRITE(21,1002) (HOLD(IM), IM = 1,M)
C WRITE(21,*) ' '
C
C CONTINUE THIS LOOP UNTIL ALL SEQUENCERS ALL FULLY SCHEDULED *********
C
100 CONTINUE
C
C ********************************************************************
C ********************************************************************
C
C PART ONE OF THE HEURISTIC ALGORITHM IS COMPLETED. EACH SEQUENCER HAS
C   AN INITIAL ASSIGNMENT OF PACK TYPES FOR ONE PRODUCTION RUN.
C
C ********************************************************************
C ********************************************************************
C
C PART TWO OF THE HEURISTIC ALGORITHM -
C   SCHEDULES ANY REMAINING UNASSIGNED PACK TYPES
C
C ********************************************************************
C ********************************************************************
C
C SELECT THE SEQUENCER WITH THE SMALLEST CUMULATIVE PVRk,for k= 1 to L,
C   TO MAKE MULTIPLE PRODUCTION RUNS **********************************
C
6000 KEY = 1
LIGHT = LOAD(1)
DO 200 IL = 2,L
   IF(LOAD(IL) .LT. LIGHT) THEN
      KEY = IL
      LIGHT = LOAD(IL)
   ENDIF
200 CONTINUE
C
```

```
C NOTE THE SEQUENCER SELECTED, AND IT'S CORESPONDING PVRk
C WRITE(21,1006) KEY, LIGHT
C
C RECORD THE NUMBER OF CHANGE-OVERS MADE BY THE kth SEQUENCER ******
C
OHAUL(KEY) = OHAUL(KEY) + 1
C
C SELECT THE FIRST PACK TYPE FOR ASSIGNMENT ON THE kth SEQUENCER
C   USING THE 'MOST COMMON' RULE *************************************
C
SMALL = 500
DO 205 IM = 1,M
C
   CALL NEXPAC(PACK(IM),DIFFER(M+KEY,IM),SMALL,INDEX,
     +                  MOST,B(IM),IM)
C
205 CONTINUE
SMALL = B(INDEX)
C
C THE FIRST PACK TYPE IS SELECTED **********************************
C
C NOTE THE PACK TYPE SELECTED, AND IT'S CORRESPONDING Bi
C WRITE(21,1005) INDEX, SMALL, MOST
C
C ASSIGN THE PACK TYPE TO THE kth SEQUENCER AND UPDATE PVRk ***********
C
CALL ASSIGN(VOLUME(INDEX),ORDER(INDEX),LOAD(KEY),SUM,
     +     ALCATE(KEY,INDEX),PACK(INDEX),TRACK(KEY,OHAUL(KEY)+1,INDEX))
PACK2(INDEX) = .FALSE.
C
C DETERMINE THE COMPONENT TYPES REQUIRED FOR THE ASSIGNED PACK TYPE ***
C
DO 207 IN = 1,N
   IF(A(INDEX,IN) .EQ. 1)  COTYPE(KEY,OHAUL(KEY)+1,IN) = 1
207 CONTINUE
C
C CHECK IF ALL PACKS ARE ASSIGNED **********************************
IF(SUM .EQ. M) GO TO 9000
C
C INITIALIZE ARRAY 'UNION'
DO 212 IN = 1,N
   UNION(IN) = 0
212 CONTINUE
C
C UPDATE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk) *******************
C
REMAIN(KEY) = HEADS(KEY) - SMALL
C
C NOTE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk)
C WRITE(21,1010) KEY, REMAIN(KEY)
C
```

```
C UPDATE THE APPROPRIATE M+kth ROW OF 'DIFFER' ************************
C
DO 210 IM = 1,M
   IF(PACK(IM)) THEN
      DIFFER(M+KEY,IM) = DIFFER(INDEX,IM)
   ELSE
      DIFFER(M+KEY,IM) = 500
   ENDIF
210  CONTINUE
C
C NOTE THE UPDATED M+kth ROW OF ARRAY 'DIFFER
C WRITE(21,*) 'DIFFER:'
C WRITE(21,1002) (DIFFER(M+KEY,IM), IM = 1,M)
C WRITE(21,*) ' '
C
C LOCATE THE 'MOST SIMILAR' UNASSIGNED PACK TYPE *********************
C
7000 SMALL = 500
DO 215 IM = 1,M
C
   CALL NEXPAC(PACK(IM),DIFFER(M+KEY,IM),SMALL,INDEX,
     +                 MOST,B(IM),IM)
C
215 CONTINUE
C
C NOTE THE PACK TYPE SELECTED, AND IT'S CORRESPONDING Bi
C WRITE(21,1005) INDEX, SMALL, MOST
C
C CHECK IF THIS UNASSIGNED PACK TYPE CAN BE ASSIGNED GIVEN THE
C   CURRENT Hk *****************************************************
C
IF((REMAIN(KEY) - SMALL) .GE. 0) THEN
C
C ASSIGN THE PACK TYPE TO THE kth SEQUENCER AND UPDATE PVRk ***********
C
   CALL ASSIGN(VOLUME(INDEX),ORDER(INDEX),LOAD(KEY),SUM,
     +     ALCATE(KEY,INDEX),PACK(INDEX),TRACK(KEY,OHAUL(KEY)+1,INDEX))
   PACK2(INDEX) = .FALSE.
   DIFFER(M+KEY,INDEX) = 500
C
C UPDATE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk) ********************
C
REMAIN(KEY) = HEADS(KEY) - SMALL
C
C NOTE THE NUMBER OF UNALLOCATED HEADS (Ck - Hk)
C WRITE(21,1010) KEY, REMAIN(KEY)
C
C PERFORM THE COMPARISON OPERATION ************************************
C
   DO 220 IM = 1,M
               IF(.NOT. PACK2(IM)) THEN
```

```
         IF(ALCATE(KEY,IM) .EQ. 1) THEN
            DO 225 IN = 1,N
C
               CALL COMPARE(A(IM,IN),UNION(IN),
     +                                  COTYPE(KEY,OHAUL(KEY)+1,IN))
C
225      CONTINUE
                  ÉNDIF
      ENDIF
220     CONTINUE
C
C CHECK IF ALL PACKS ARE ASSIGNED ************************************
   IF(SUM .EQ. M) GO TO 9000
C
C NOTE THE 'U' VECTOR
C    WRITE(21,*) 'ARRAY UNION:'
C    WRITE(21,1003) (UNION(IN), IN = 1,N)
C
   DO 230 IM = 1,M
      IF(PACK(IM)) THEN
 DIFFER(M+KEY,IM) = 0
 DO 235 IN = 1,N
C
            CALL NEWDIF(COMP(IN),A(IM,IN),UNION(IN),
     +                            DIFFER(M+KEY,IM))
C
235  CONTINUE
      ENDIF
230     CONTINUE
C
C COMPARISON OPERATION IS COMPLETE ***********************************
C
C NOTE THE UPDATED M+kth ROW OF ARRAY 'DIFFER'
C    WRITE(21,*) ' '
C    WRITE(21,*) 'DIFFER:'
C    WRITE(21,1002) (DIFFER(M+KEY,IM), IM = 1,M)
C    WRITE(21,*) ' '
C
C CONTINUE THIS LOOP UNTIL EITHER Ck PREVENTS ANY OTHER PACK TYPES
C   FROM BEING ASSIGNED, OR UNTIL ALL PACK TYPES ARE ASSIGNED *********
C
   GO TO 7000
ELSE
   DO 240 IM = 1,M
      PACK2(IM) = .TRUE.
240     CONTINUE
   GO TO 6000
ENDIF
C
C STOP THE CPU TIMER
9000 IF(.NOT. LIB$SHOW_TIMER(INITIM)) GO TO 9900
```

```
C
C ********************************************************************
C ********************************************************************
C
C PART TWO OF THE HEURISTIC ALGORITHM IS COMPLETED. ALL PACK TYPES
C   ARE SCHEDULED FOR PRODUCTION
C
C ********************************************************************
C ********************************************************************
C
C PRINT THE INPUT PARAMETERS, KNOWN DATA, DEVISED PRODUCTION SCHEDULE,
C   AND ASSOCIATED PRODUCTION SCHEDULE CHARACTERISTICS.
C
WRITE(21,1001) L, M, N
WRITE(21,1019) VOLTOT, EQUAL
WRITE(21,*) 'ARRAY V:'
WRITE(21,1002) (V(IM), IM = 1,M)
WRITE(21,*) ' '
WRITE(21,*) 'ARRAY HEADS:'
WRITE(21,1002) (HEADS(IL), IL = 1,L)
WRITE(21,*) ' '
WRITE(21,*) 'ARRAY ORDER:'
WRITE(21,1002) (ORDER(IM), IM = 1,M)
WRITE(21,*) ' '
WRITE(21,*) 'ARRAY B:'
WRITE(21,1002) (B(IM), IM = 1,M)
WRITE(21,*) ' '
WRITE(21,*) 'ARRAY TOTAL:'
WRITE(21,1004) (TOTAL(IM), IM = 1,M)
WRITE(21,*) ' '
WRITE(21,*) 'ARRAY VOLUME:'
WRITE(21,1004) (VOLUME(IM), IM = 1,M)
WRITE(21,*) ' '
WRITE(21,*) 'TOTAL WORKLOAD PER SEQUENCER IS:'
WRITE(21,1004) (LOAD(IL), IL = 1,L)
C LOOK AT ARRAY 'DIFFER' IF SO DESIRED
C MAX = M + L
C WRITE(21,*) ' '
C WRITE(21,*) 'ARRAY DIFFER:'
C DO 902 IM = 1,MAX
C         WRITE(21,1002) (DIFFER(IM,JM), JM = 1,M)
C902 CONTINUE
WRITE(21,*) ' '
C
C PRINT THE TOTAL NUMBER OF SEQUENCED TAPES PRODUCED (SUM) ************
WRITE(21,1008) SUM
WRITE(21,*)'THE TOTAL NUMBER OF CHANGE-OVERS PER SEQUENCER IS:'
WRITE(21,1002) (OHAUL(IL), IL = 1,L)
C
C PRINT THE PACK TYPES ASSIGNED TO EACH SEQUENCER FOR EACH RUN ********
DO 260 IL = 1,L
```

```
   MAXRUN = OHAUL(IL) + 1
   DO 265 JM = 1,MAXRUN
      WRITE(21,1013) IL,JM
      WRITE(21,1014) (TRACK(IL,JM,IM),IM = 1,M)
265    CONTINUE
260 CONTINUE
C
C PRINT THE COMPONENT TYPES REQUIRED BY EACH SEQUENCER FOR EACH RUN ***
DO 300 IL = 1,L
   MAXRUN = OHAUL(IL) + 1
   DO 305 JM = 1,MAXRUN
      COUNT = 0
      DO 310 IN = 1,N
 IF(COTYPE(IL,JM,IN) .EQ. 1) THEN
    COUNT = COUNT + 1
    TYPE(COUNT) = IN
        ENDIF
C  RE-INITIALIZE ARRAY 'UNION' FOR LATER OUTPUT
         UNION(IN) = 0
310       CONTINUE
      SPACES(IL,JM) = COUNT
      WRITE(21,1016) IL, JM, COUNT
      WRITE(21,1017) (TYPE(IN), IN = 1,COUNT)
305    CONTINUE
300 CONTINUE
C
C PRINT ALL OF THE COMPONENT TYPES REQUIRED BY EACH SEQUENCER TO
C   PRODUCE ALL OF THEIR ASSIGNED SEQUENCED TAPES ********************
DO 320 IL = 1,L
   COUNT = 0
   DO 325 IM = 1,M
      IF(ALCATE(IL,IM) .EQ. 1) THEN
         DO 330 IN = 1,N
    IF(A(IM,IN) .EQ. 1)  UNION(IN) = 1
330          CONTINUE
      ENDIF
325    CONTINUE
   DO 335 IN = 1,N
      IF(UNION(IN) .EQ. 1) THEN
 COUNT = COUNT + 1
 TYPE(COUNT) = IN
      ENDIF
C  RE-INITIALIZE ARRAY 'UNION' FOR LATER OUTPUT
      UNION(IN) = 0
335    CONTINUE
   SEQTOT(IL) = COUNT
   WRITE(21,1018) IL, COUNT
   WRITE(21,1017) (TYPE(IN), IN = 1,COUNT)
320 CONTINUE
C
DO 350 IL = 1,L
```

```
    MAXRUN = OHAUL(IL) + 1
    INTSEC(IL) = 0
    OTHER(IL,1) = 0
           IF(MAXRUN .EQ. 2) THEN
       SMALL = SPACES(IL,1)
       IF(SPACES(IL,2) .LE. SMALL)  SMALL = SPACES(IL,2)
       DO 355 IN = 1,N
  IF((COTYPE(IL,1,IN) + COTYPE(IL,2,IN)) .EQ. 2) THEN
     INTSEC(IL) = INTSEC(IL) + 1
     SMALL = SMALL - 1
  ENDIF
355          CONTINUE
       OTHER(IL,2) = SMALL
       GO TO 350
    ELSE
       IF(MAXRUN .GE. 3) THEN
          DO 360 JM = 1,MAXRUN
     COUNT = 0
     DO 365 IN = 1,N
        IF(COTYPE(IL,JM,IN) .EQ. 1)UNION(IN)=UNION(IN)+1
        IF(JM .LE. (MAXRUN-1)) THEN
           IF(COTYPE(IL,JM,IN) + COTYPE(IL,JM+1,IN)
      +                                                      .EQ. 2) THEN
      COUNT = COUNT + 1
      OTHER(IL,JM+1) = SPACES(IL,JM+1) - COUNT
                             ENDIF
        ENDIF
365            CONTINUE
360          CONTINUE
          DO 370 IN = 1,N
     IF(UNION(IN) .EQ. MAXRUN) INTSEC(IL)=INTSEC(IL) + 1
     UNION(IN) = 0
370           CONTINUE
      ENDIF
    ENDIF
350 CONTINUE
C
C PRINT THE NUMBER OF DEDICATED HEADS ON EACH SEQUENCER ***************
DO 375 IL = 1,L
   IF(OHAUL(IL) .EQ. 0) THEN
      WRITE(21,1024) IL
   ELSE
      WRITE(21,1020) IL, INTSEC(IL), IL,
     +                   FLOAT(INTSEC(IL))/FLOAT(HEADS(IL))
   ENDIF
375 CONTINUE
C
C PRINT THE NUMBER OF COMPONENT TAPE CHANGES REQUIRED BY EACH
C   SEQUENCER BETWEEN CONSECUTIVE PRODUCTION RUNS *********************
DO 385 IL = 1,L
   MAXRUN = OHAUL(IL) + 1
   IF(MAXRUN .EQ. 1) THEN
      WRITE(21,1023) IL
   ELSE
      DO 390 JM = 2,MAXRUN
         WRITE(21,1022) IL, JM, OTHER(IL,JM)
390        CONTINUE
   ENDIF
385 CONTINUE
C
C PRINT THE TOTAL NUMBER OF COMPONENT TAPE CHANGES REQUIRED BY
C   EACH SEQUENCER OVER ALL PRODUCTION RUNS ***************************
DO 400 IL = 1,L
   COUNT = 0
   MAXRUN = OHAUL(IL) + 1
   IF(MAXRUN .GE. 2) THEN
      DO 405 JM = 2,MAXRUN
 COUNT = COUNT + OTHER(IL,JM)
405        CONTINUE
      WRITE(21,1021) IL, COUNT
   ENDIF
400 CONTINUE
C
C
1000 FORMAT (I4,I5,I5)
1001    FORMAT ('0', 'YFIX:  L=',I2, 3X, 'M=',I3, 3X, 'N=',I4, /)
1002 FORMAT (12(1X,I4))
1003 FORMAT (30(1X,I1))
1004 FORMAT (9(1X,I6))
1005 FORMAT (' ', 'INDEX:',I4, 3X, 'SMALL:',I4, 3X, 'MOST:',I4,/)
1006    FORMAT (' ', 'KEY:', I2, 3X, 'LIGHT: ', I6, /)
1007 FORMAT (' ',20(1X,L2))
1008 FORMAT (' TOTAL NUMBER OF SEQUENCED TAPES PRODUCED IS: ',I3,/)
C 1009 FORMAT ('0', 'PACK',I4)
1010 FORMAT (' ','SEQUENCER# IS:',I3,6X,'#HEADS REMAINING IS:',I4,/)
1011 FORMAT (' ', 'Y',I1,',', I3,'  = 1')
1013 FORMAT ('0','SEQUENCER#',I3,4X,'RUN#',I4,/,' PACK TYPES
    +ASSIGNED ARE:')
1014 FORMAT (15(1X,I3))
1016    FORMAT ('0', 'FOR SEQUENCER#',I3,3X,'RUN#',I4,/,
    + ' TOTAL # OF COMPONENT TYPES REQUIRED =',I4,/,
    + ' COMPONENT TYPES REQUIRED ARE:')
1017 FORMAT (15(1X,I3))
1018 FORMAT ('0', 'FOR SEQUENCER#',I3,/,
    + ' TOTAL # OF COMPONENT TYPES REQUIRED FOR ALL RUNS IS:',I4,/,
    + ' COMPONENT TYPES REQUIRED ARE:')
1019 FORMAT (' ', 'TOTAL VOLUME OF ALL SEQUENCERS =',I7,//,
    + ' MAXIMUM LOAD PERMITTED ON ANY SEQUENCER DURING ITS
    +FIRST RUN = ',I7,/)
1020 FORMAT ('0', 'THE NUMBER OF DISPENSING HEADS DEDICATED ON
    + SEQUENCER# ',I1,' = ',I3,//, ' THE PROPORTION OF DEDICATED HEADS
    + ON SEQUENCER# ',I1,' = ',F5.3)
```

```
1021 FORMAT ('0', 'THE TOTAL NUMBER OF COMPONENT TAPE CHANGES
     + REQUIRED BY SEQUENCER# ',I1,' TO PRODUCE
     + ALL OF ITS ASSIGNED SEQUENCED TAPES IS: ',I3)
1022 FORMAT ('0', 'THE NUMBER OF COMPONENT TAPES THAT NEED
     + TO BE CHANGED ON',/,
     +' SEQUENCER# ',I1,' TO PRODUCE RUN# ',I1,' = ',I3)
1023 FORMAT ('0', 'SEQUENCER# ',I1,' REQUIRES NO COMPONENT TAPE
     + CHANGES SINCE IT IS ONLY MAKING ONE RUN')
1024 FORMAT ('0', 'SEQUENCER# ',I1,' IS MAKING ONLY ONE RUN;',/,
     +' THEREFORE, ALL OF ITS DISPENSING HEADS ARE DEDICATED')
C
C CLOSE OUTPUT FILE
CLOSE(UNIT=21)
C
STOP
9900 END
C
C ***********************************************************************
C ***********************************************************************
C
C END OF MAIN PROGRAM
C
C ***********************************************************************
C ***********************************************************************
C
C PROGRAM SUBROUTINES -
C   ALL VARIABLES IN THE SUBROUTINES HAVE THE SAME DEFINITION AS IN
C   THE MAIN PROGRAM
C
C ***********************************************************************
        SUBROUTINE RESULT (INDEX,IM,ITEMP,SMALL,MOST,B)
C ***********************************************************************
C
INTEGER INDEX, IM, ITEMP, SMALL, MOST, B
C
INDEX = IM
ITEMP = INDEX
MOST = B - SMALL
C
RETURN
END
C
C ***********************************************************************
SUBROUTINE ASSIGN (VOLUME,ORDER,LOAD,SUM,ALCATE,PACK,TRACK)
C ***********************************************************************
C
INTEGER VOLUME, ORDER, LOAD, SUM, ALCATE, TRACK
LOGICAL PACK
C
   ALCATE = 1
   PACK = .FALSE.
   LOAD = LOAD + VOLUME
   TRACK = ORDER
   SUM = SUM + 1
C
RETURN
END
C
C *******************************************************************
SUBROUTINE NEXPAC (PACK,DIFFER,SMALL,INDEX,MOST,B,IM)
C *******************************************************************
C
INTEGER DIFFER, SMALL, INDEX, MOST, B, IM
LOGICAL PACK
C
IF(PACK) THEN
   IF(DIFFER .LT. SMALL) THEN
      INDEX = IM
      SMALL = DIFFER
              MOST = B - SMALL
   ELSE
      IF(DIFFER .EQ. SMALL .AND. (B - SMALL) .GT. MOST) THEN
           INDEX = IM
 MOST = B - SMALL
      ENDIF
   ENDIF
ENDIF
C
RETURN
END
C
C *******************************************************************
SUBROUTINE COMPARE (A,UNION,COTYPE)
C *******************************************************************
C
INTEGER A, UNION, COTYPE
C
        IF(A .EQ. 1) THEN
   UNION = 1
           COTYPE = 1
ENDIF
C
RETURN
END
C
C *******************************************************************
SUBROUTINE NEWDIF (COMP,A,UNION,DIFFER)
C *******************************************************************
C
INTEGER COMP, A, UNION, DIFFER
C
COMP = A
```